Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Desarrollo de una plataforma para experimentación con Interface Cerebro Ordenador (BCI)

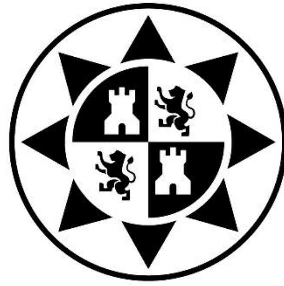
TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Irene Pérez Aguirre**
Director: José Manuel Cano Izquierdo
Codirector: Miguel Almonacid Kroeger

Cartagena, septiembre de 2017

Universidad Politécnica de Cartagena



UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Departamento de Ingeniería de Sistemas y Automática

DESARROLLO DE UNA PLATAFORMA PARA EXPERIMENTACIÓN CON INTERFACE CEREBRO ORDENADOR (BCI)

Trabajo Fin de Grado

Irene Pérez Aguirre

Director: José Manuel Cano Izquierdo

Codirector: Miguel Almonacid Kroeger

2017

A mi Madre y mi Padre, por enseñarme que la suerte solo existe a ratos, y que hay que creer en la Justicia y el Esfuerzo.

A mi Hermano, mi Abuela y mi Tita, por apoyarme siempre y admirar mi trabajo.

A Manuel, por recordarme en todo momento que soy capaz y ser mi gran apoyo.

A mi Consejo de Sabios y demás amigos, por ser compañeros, no solo de estudio sino también de vida estos años.

A tantos profesores que me han ayudado a mejorar, fomentando mi interés por las ciencias y la ingeniería.

Resumen

Actualmente, el desarrollo tecnológico que se está experimentando ha permitido ampliar el campo de aplicación de la robótica. Respecto al ámbito de la Ingeniería Biomédica, se han creado robots de asistencia a personas paralíticas, robots guía para personas invidentes y prótesis robóticas, entre otras. Sin embargo, al centrarse en las aplicaciones que incluyen robots controlados mentalmente mediante señales cerebrales, no son tantos los ejemplos que se podrían mencionar.

Por esta razón, en este Trabajo Fin de Grado se presenta una plataforma para experimentación con Interface Cerebro Ordenador (BCI), que podría sentar una base sobre la que desarrollar un robot basado en esta tecnología para personas con discapacidad neuromuscular severa. Para ello, se ha considerado necesario diseñar una interfaz de interacción hombre-máquina que incluya una aplicación robótica diseñada en lazo cerrado para proporcionar una realimentación al usuario. De esta forma, se podrá demostrar con pruebas el funcionamiento adecuado de un sistema que puede servir como preámbulo para sucesivas investigaciones.

Abstract

Nowadays, technological development has led to widening robotic application field. Regarding Biomedical Engineering, different types of robots have been created, such us assistive robots for disabled people, guide robots for blind people, and even robotic prosthetics, among others. However, if we try to focus on mind controlled robots, using brain signals, there are not a great number of examples to quote.

Because of this fact, a Brain-Computer Interfaced (BCI) experimentation platform is introduced in this Final Project, which could lay the foundations for new research in this field. Consequently, it has been necessary to design a human-machine interaction interface which includes a closed-loop robotic application to provide feedback to the user. In this way, it could be demonstrated the suitable system operation, which could be used as a preamble for consecutive works.

Índice de Contenido

Capítulo 1 .... **19**

Introducción **19**

1.1 Marco de trabajo 19

1.2 Motivación 20

1.3 Descripción 21

1.4 Objetivos 23

Capítulo 2 .... **25**

Estado del arte **25**

2.1 Introducción: ¿qué es el BCI? 25

2.2 Fundamentos del BCI 28

2.2.1 Señales cerebrales 28

2.2.2 Medición de las señales cerebrales 29

2.3 Evolución del BCI 32

2.4 Curvas de expectación de Gartner ... 36
2.4.1 Introducción ... 36
2.4.2 Aplicación al BCI ... 39
2.5 Hardware disponible ... 41
2.5.1 BCI ... 41
2.5.2 Aplicación robótica ... 47
2.6 Software disponible ... 50
2.6.1 Orca ... 50
2.6.2 YARP ... 50
2.6.3 MIRA ... 51
2.6.4 ROS ... 51

Capítulo 3 ... 53
Emotiv Insight ... 53
3.1 Introducción ... 53
3.2 Adquisición del dispositivo ... 53
3.3 Descripción del hardware ... 54
3.3.1 Especificaciones técnicas ... 56
3.3.2 Puesta en marcha ... 57
3.3.3 Colocación del casco ... 61
3.3.4 Detección ... 64
3.4 Descripción del software ... 65
3.4.1 Emotiv Control Panel ... 65
3.4.2 Application ... 67
3.4.3 Connect ... 67

3.4.4 Set Up ... 68
3.4.5 Detections ... 68
3.4.6 Recording ... 79
3.4.7 Reports ... 79
3.4.8 Help ... 80
3.5 Emotiv Composer ... 81

Capítulo 4 ... 87

Kinova Mico² ... 87

4.1 Introducción ... 87
4.2 Descripción del hardware ... 87
4.2.1 Especificaciones técnicas ... 87
4.2.2 Puesta en marcha ... 90
4.2.3 Principios de operación ... 93
4.3 Descripción del software ... 95
4.3.1 ROS ... 95
4.3.2 Herramientas software ... 98

Capítulo 5 ... 101

Preparación del entorno y puesta en marcha ... 101

5.1 Introducción ... 101
5.2 Preparación del software ... 102
5.2.1 Ubuntu ... 102
5.2.2 SDK Emotiv ... 103
5.2.3 Kinova-api ... 105

5.2.4 Code::Blocks 106
5.2.5 ROS 108
5.3 Preparación del hardware 114
5.3.1 Kinova Mico [2] 114
5.3.2 Emotiv Insight 114
5.4 Puesta en marcha 114
5.4.1 Kinova Mico [2] 115
5.4.2 Emotiv Insight 118

Capítulo 6 119

Software desarrollado 119

6.1 Introducción 119
6.2 Kinova Mico [2] 119
6.2.1 Simulación 120
6.2.2 Real 127
6.3 Emotiv Insight 132
6.3.1 Main 133
6.3.2 Funciones 138

Capítulo 7 143

Funcionamiento de la aplicación 143

7.1 Introducción 143
7.2 Puesta en marcha 144
7.3 Interfaz gráfica 150

Capítulo 8 .... **155**

Pruebas y resultados finales **155**

8.1 Introducción 155

8.2 Pruebas realizadas 157

8.2.1 Kinova y simulación de Emotiv 157

8.2.2 Kinova y Emotiv 158

8.2.3 Entrenamiento y comprobación para 4 sujetos 159

Capítulo 9 .... **163**

Conclusiones y trabajos futuros **163**

9.1 Conclusiones 163

9.2 Trabajos futuros 167

Anexo I **171**

Creación de un espacio de trabajo en ROS **171**

Anexo II **173**

Código de programación de Kinova para Gazebo **173**

Anexo III **179**

Código de programación de Kinova **179**

Anexo IV **185**

Código de programación de Emotiv Insight **185**

Trabajos citados **197**

Índice de Figuras

Capítulo 1

Figura 1.1 Esquema básico de la arquitectura del sistema 21

Figura 1.2 Flujograma del funcionamiento de la aplicación......................... 22

Capítulo 2

Figura 2.1 Representación esquemática de una aplicación BCI 26

Figura 2.2 BCI en lazo abierto 27

Figura 2.3 BCI en lazo cerrado 27

Figura 2.4 Zonas de colocación de sensores 29

Figura 2.5 Electroencefalograma 29

Figura 2.6 Electrocorticograma 30

Figura 2.7 Array multielectrodo intracortical 30

Figura 2.8 Escáner fMRI de Siemens (MAGNETOM® Spectra)................. 31

Figura 2.9 Imagen fMRI con áreas amarillas mostrando el incremento de la actividad cerebral comparada con la condición de control........................... 31

Figura 2.10 Escáner MEG de Elekta (Neuromag® TRIUX)........................ 31

Figura 2.11 Imagen MEG mostrando la intensidad de la actividad cerebral31

Figura 2.12 Electrómetro de capilaridad de Lippmann................................ 33

Figura 2.13 Galvanómetro de Einhoven... 34

Figura 2.14 Grabación EEG realizada por Berger 35
Figura 2.15 Esquema del experimento de Fetz con monos 35
Figura 2.16 Fases del ciclo de vida de una tecnología.................................. 37
Figura 2.17 Esquema de las zonas de las curvas de expectación de Gartner38
Figura 2.18 Curva de Gartner 2013 39
Figura 2.19 Curva de Gartner 2014 40
Figura 2.20 Curva de Gartner 2015 40
Figura 2.21 Curva de Gartner 2016 41
Figura 2.22 Emotiv Insight 42
Figura 2.23 Emotiv EPOC+ 43
Figura 2.24 OpenBCI. (a)Casco Ultracortex "Mark IV" EEG (b)Cyton Biosensing de 8 canales 43
Figura 2.25 Neurosky MindWave Mobile EEG.. 44
Figura 2.26 BrainProducts actiCAP Xpress 45
Figura 2.27 Olimex EEG-SMT... 46
Figura 2.28 InterAxon Muse 46
Figura 2.29 Carrera de drones BCI 47
Figura 2.30 Control de LEGO Mindstorms NXT con BCI.......................... 48
Figura 2.31 Control de mano robótica con BCI... 48
Figura 2.32 Shadow Dexterous Hand... 49
Figura 2.33 Kinova Mico²... 49

Capítulo 3

Figura 3.1 Página Web de Emotiv Insight... 54
Figura 3.2 Ensamblaje de Emotiv Insight.. 55
Figura 3.3 Contenido de un pack de sensores 55
Figura 3.4 Cargador del Emotiv Insight 56
Figura 3.5 Receptor Universal USB para Bluetooth SMART 56
Figura 3.6 Significado del indicador luminoso del receptor USB 58
Figura 3.7 Indicadores luminosos del receptor USB..................................... 59

Figura 3.8 Vista principal de la aplicación ControlPanel 60
Figura 3.9 Vista Insight de la aplicación ControlPanel 60
Figura 3.10 Indicadores de conexión y carga 60
Figura 3.11 Comprobación de la buena conexión de los electrodos 61
Figura 3.12 Electrodos de referencia 62
Figura 3.13 Electrodos AF3 y AF4 62
Figura 3.14 Electrodo T7 63
Figura 3.15 Electrodo T8 63
Figura 3.16 Electrodo Pz... 63
Figura 3.17 Pantalla principal del Emotiv ControlPanel............................. 66
Figura 3.18 Menú del Emotiv ControlPanel 66
Figura 3.19 Menú del Emotiv ControlPanel: "Application" 67
Figura 3.20 Menú del Emotiv ControlPanel: "Connect" 67
Figura 3.21 Menú del Emotiv ControlPanel: "Detections"........................... 68
Figura 3.22 Cubo 3D para realimentación 69
Figura 3.23 Pantalla de acción de Comando Mental 70
Figura 3.24 Pantalla de entrenamiento de Comando Mental 71
Figura 3.25 Grabación del estado neutral 73
Figura 3.26 Pantalla de configuración avanzada de Comando Mental 74
Figura 3.27 Pantalla de ajustes de Comando Mental 75
Figura 3.28 Importar modelo para entrenamiento 75
Figura 3.29 Avatar para realimentación 76
Figura 3.30 Pantalla de sensibilidad de Expresiones Faciales...................... 77
Figura 3.31 Pantalla de entrenamiento de Expresiones Faciales 78
Figura 3.32 Gráficas de Estados de Ánimo 79
Figura 3.33 Menú del Emotiv ControlPanel: "Help" 80
Figura 3.34 Información sobre Emotiv ControlPanel................................... 80
Figura 3.35 Pantalla Interactiva de Emotiv Composer................................ 81
Figura 3.36 Pantalla EmoScript de Emotiv Composer 82
Figura 3.37 Calidad de contacto en Emotiv Composer................................ 84

Capítulo 4

Figura 4.1 Partes del Mico^2 88
Figura 4.2 Conexiones del Mico^2 89
Figura 4.3 Sujeción del brazo a la mesa 90
Figura 4.4 Alimentación del Mico^2 91
Figura 4.5 Conexión de Mico^2 con batería 91
Figura 4.6 Conexión del último actuador 92
Figura 4.7 Conexión de la mano 92
Figura 4.8 Joystick del Mico^2 93
Figura 4.9 Posición home 94
Figura 4.10 Posición retracted 94
Figura 4.11 Esquema básico del envío de mensajes ROS 97
Figura 4.12 Icono de Gazebo 98
Figura 4.13 Icono de Code::Blocks 99
Figura 4.14 Icono de CMake 99
Figura 4.15 Icono de wxWidgets 100

Capítulo 5

Figura 5.1 Mensaje de instalación de Kinova-api 105
Figura 5.2 Instalación exitosa de Kinova-api 105
Figura 5.3 Añadir ruta de include en Codeblocks 106
Figura 5.4 Añadir C++11 en Codeblocks 107
Figura 5.5 Añadir ruta de librerías en Codeblocks 107
Figura 5.6 Añadir ruta de directorios en Codeblocks 108
Figura 5.7 Contenido de kinova-ros 109
Figura 5.8 Encabezado kinova_robot.launch 111
Figura 5.9 Descripción del nodo para kinova-control 111
Figura 5.10 Contenido del CMakeLists.txt de kinova-control 112
Figura 5.11 Configuración de kinova-gazebo 112

Figura 5.12 Conexión de Kinova al ordenador 115
Figura 5.13 Lanzamiento de Roslaunch 116
Figura 5.14 Sistema de coordenadas del brazo robótico............................ 117

Capítulo 6

Figura 6.1 Main del programa para Gazebo 120
Figura 6.2 Función argumentParser del programa para Gazebo 122
Figura 6.3 Función moveJoint del programa de Gazebo 123
Figura 6.4 Función listener del programa de Gazebo 124
Figura 6.5 Función callback del programa de Gazebo 125
Figura 6.6 Función moveFingers del programa de Gazebo 126
Figura 6.7 Cabecera del programa de Gazebo 127
Figura 6.8 Main del programa de Kinova 127
Figura 6.9 Función argumentParser() del programa de Kinova 128
Figura 6.10 Función homeRobot() del programa de Kinova 128
Figura 6.11 Función cartesian_pose_client() del programa de Kinova..... 129
Figura 6.12 Función gripper_client() del programa de Kinova 130
Figura 6.13 Operaciones con cuaternios en programa de Kinova 130
Figura 6.14 Fragmento de callback de programa de Kinova 131
Figura 6.15 Cabeceras del programa de Kinova.. 131
Figura 6.16 Flujograma de funcionamiento de EmoEngine 132
Figura 6.17 Declaración de variables en programa de Emotiv.................. 133
Figura 6.18 Inicio de ROS y EmoEngine en programa de Emotiv 134
Figura 6.19 Configuración en programa de Emotiv 135
Figura 6.20 Proceso tras detección de EmoState en programa de Emotiv 136
Figura 6.21 Detección de "sorpresa" en programa de Emotiv 136
Figura 6.22 Control del envío de mensajes en programa de Emotiv.......... 137
Figura 6.23 Desconexión de EmoEngine en programa de Emotiv 137
Figura 6.24 Conexión en programa de Emotiv 138
Figura 6.25 Función signal_handler en programa de Emotiv 139

Figura 6.26 Inicio de sesión en programa de Emotiv 140
Figura 6.27 Información del tiempo y la conexión en programa de Emotiv 141
Figura 6.28 Información de calidad del contacto en programa de Emotiv 141

Capítulo 7

Figura 7.1 Dispositivos de la aplicación 143
Figura 7.2 Establecer conexión con el robot 144
Figura 7.3 Dirección IP del PC1 145
Figura 7.4 Dirección IP del PC2 145
Figura 7.5 Variable de entorno ROS_MASTER_URI 146
Figura 7.6 Ejecución del Roscore 147
Figura 7.7 Acceso al ssh Listener 147
Figura 7.8 Acceso al ssh del Talker 148
Figura 7.9 Interfaz para Kinova 150
Figura 7.10 Interfaz para Emotiv 151

Capítulo 8

Figura 8.1 Tiempo, calidad de señal y de contacto en el log 156
Figura 8.2 Detección y potencia de la acción en el log 156

Capítulo 9

Figura 9.1 Pregunta en GitHub sobre problema del SDK 168

Índice de Tablas

Capítulo 3

Tabla 3.1 Especificaciones técnicas de Emotiv Insight 57

Tabla 3.2 Expresiones faciales y comandos mentales................................... 65

Tabla 3.3 Estados emocionales... 65

Capítulo 4

Tabla 4.1 Especificaciones técnicas generales del Mico^2............................... 88

Tabla 4.2 Especificaciones de comunicación del Mico^2................................. 89

Tabla 4.3 Especificaciones técnicas de la mano 90

Capítulo 8

Tabla 8.1 Comparativa de sujetos en experimento BCI 161

Capítulo 1

Introducción

1.1 Marco de trabajo

El presente proyecto tiene como objetivo principal el desarrollo de una aplicación física que integre un elemento *hardware* (robot) con un sistema Interfaz Cerebro Ordenador (BCI, por sus siglas en inglés *Brain Computer Interface*). Por tanto, mantiene una estrecha relación con la línea de investigación de BCI seguida en el Departamento de Ingeniería de Sistemas y Automática de la Universidad Politécnica de Cartagena (UPCT).

Sin embargo, aunque el proyecto *per se* no pertenece a dicha línea de investigación, supone un avance en la misma. Esto se debe a que, previamente a este Trabajo Fin de Grado (TFG), la aplicación del BCI en el departamento se había limitado al estudio y tratamiento de las señales cerebrales, pero no al uso de las mismas para controlar un determinado sistema robótico. Además, tal y como se detalla en posteriores capítulos, se ha hecho uso de *hardware* innovador, así como del *middleware* robótico ROS (*Robot Operating System*), lo que confiere al proyecto un potencial remarcable.

1.2 Motivación

A día de hoy, no cabe duda de que la robótica es una de las ramas de la tecnología que influye de manera más directa en la evolución de la humanidad. Esto ha generado un interés económico que ha llevado a invertir un gran capital, tanto en la investigación como en el desarrollo de nuevos sistemas robóticos [1]. Por tanto, cualquier avance en el campo, por nimio que parezca, puede llegar a ser un nuevo punto de partida sobre el que seguir experimentando con un objetivo único: facilitar la vida de las personas. Ésta es, posiblemente, la principal razón del auge que está viviendo la robótica y el principal objetivo del desarrollo de este proyecto.

Por otro lado, cabe destacar el interés existente en el departamento de Ingeniería de Sistemas y Automática por la tecnología BCI. Esto es debido a que se trata de un campo de investigación de gran potencial, pero que requiere de mucho esfuerzo y dedicación tanto para la adquisición de las señales, como para su posterior tratamiento y aplicación.

Además, este departamento busca también extender el interés por el BCI entre la gente que no conoce su existencia. Es por esto que se busca una forma de facilitar la divulgación científica mediante una plataforma de experimentación que llame la atención del público, incluso fomentando su participación.

A lo largo de las últimas décadas, se ha incrementado notablemente el interés por desarrollar una interfaz de comunicación efectiva que conecte el cerebro humano con un computador. Las investigaciones llevadas a cabo se han realizado principalmente con tres objetivos [2]:

- Crear un nuevo canal de comunicación para pacientes con discapacidades neuromusculares severas.
- Hacer del BCI una poderosa herramienta de trabajo en neurociencia computacional para contribuir a un mejor entendimiento del cerebro.
- Crear una forma de comunicación genérica para interacción hombre-máquina, aplicable a distintos campos científicos y prácticos.

En este caso, el trabajo se enmarcaría en el tercer objetivo, puesto que se trata de buscar un entendimiento entre el cerebro humano y el robot, pero siempre buscando como punto final el facilitar la vida de las personas con discapacidad.

Por tanto, este TFG se plantea con el objetivo global de aplicar la tecnología BCI a la robótica, de forma que permita evaluar las posibilidades de dicha aplicación, y conocer las limitaciones de la misma.

1.3 Descripción

El sistema está compuesto por distintos elementos *hardware* y *software* que se encuentran convenientemente integrados. Estos se enumeran a continuación y se describirán con detalle en posteriores capítulos.

Hardware:

- Casco BCI Emotiv Insight
- Brazo Robótico Kinova Mico[2] con mano de 3 dedos

Software:

- Emotiv Xavier Control Panel y Emotiv Composer
- ROS y Gazebo

A continuación, en la figura 1.1, se muestra una infografía de la relación existente entre los principales elementos mencionados.

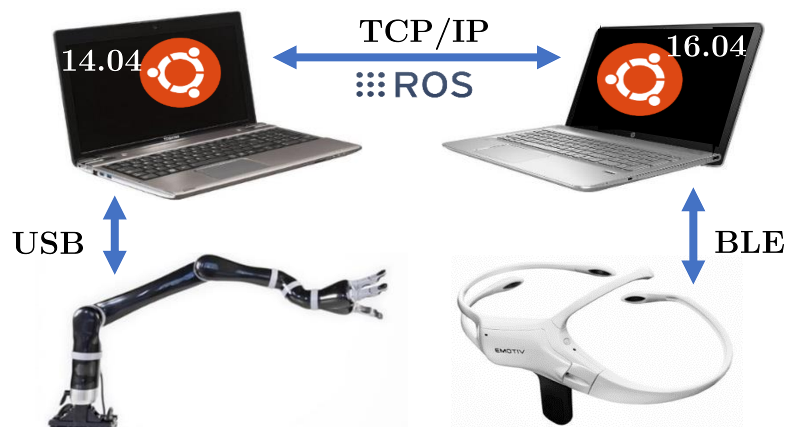


Figura 1.1 Esquema básico de la arquitectura del

Para complementar la información que se muestra en la Figura 1.2, se añade un flujograma que incluye, a grandes rasgos, la información del funcionamiento de la plataforma para experimentación con BCI.

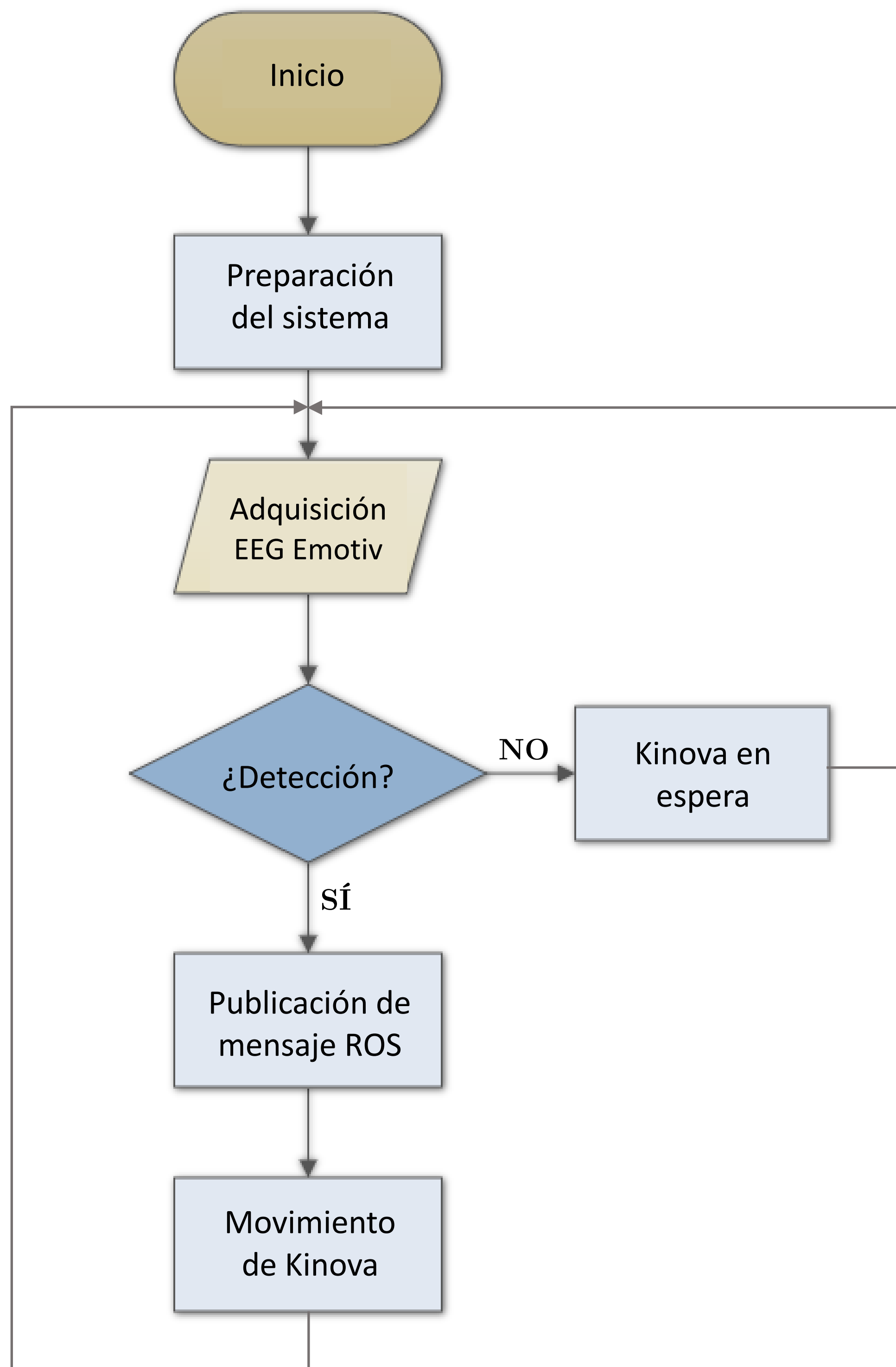


Figura 1.2 Flujograma del funcionamiento de la aplicación

Tal y como se observa en el flujograma, el proceso consta de dos partes bien diferenciadas: el robot y el casco. El casco se encarga de adquirir las señales cerebrales y realizar la detección de un estado mental. Tras ello, envía un mensaje representativo del estado al robot, que permanecía en estado de escucha mientras no le llegaban mensajes. Cuando lo recibe, lo procesa y realiza un movimiento que varía según el mensaje.

1.4 Objetivos

El objetivo final de este proyecto es el desarrollo de una plataforma para experimentación con interfaz cerebro-ordenador y, en concreto, la creación de un sistema que integre un dispositivo BCI con un robot. Para alcanzar dicho objetivo, se han cubierto las siguientes tareas de investigación y desarrollo:

- Estudio del estado del arte de las interfaces cerebro-ordenador, teniendo en cuenta sus aplicaciones médicas e ingenieriles.
- Análisis de los principios biológicos del BCI.
- Estudio de distintos cascos BCI comerciales.
- Estudio de distintos tipos de dispositivos de adquisición disponibles.
- Estudio de la evolución de la tecnología BCI mediante las curvas de expectación de Gartner.
- Elección del *software* y *hardware* a utilizar en el proyecto, así como de la aplicación concreta a desarrollar.
- Selección de los lenguajes de programación para el desarrollo software.
- Familiarización y aprendizaje de Linux, Python y wxWidget.
- Integración en ROS del Emotiv Insight.
- Integración en ROS del Kinova Mico^2.
- Desarrollo de un sistema basado en ROS distribuido en dos máquinas.
- Conexión de las dos máquinas ROS utilizadas.
- Integración de las aplicaciones ofrecidas por el fabricante del casco con las aplicaciones propias desarrolladas.

- Diseño y programación de una interfaz gráfica para facilitar el uso de la aplicación.
- Realización de pruebas mediante simulación, tanto del casco como del robot.
- Realización de pruebas en el laboratorio usando el robot y el casco.
- Realización de pruebas con varios sujetos para extraer conclusiones de las comparaciones.

Cabe destacar que muchos de los objetivos arriba enumerados no estaban incluidos en la propuesta del proyecto. Esto es debido a que son resultado de requerimientos del trabajo que han ido surgiendo y ampliaciones que se han incluido con el fin de mejorar el resultado final.

Capítulo 2

Estado del arte

2.1 Introducción: ¿qué es el BCI?

Una *Brain Computer Interface* es una vía de comunicación directa entre el cerebro de un ser vivo y un dispositivo externo.

El hecho de que se trabaje con sensores colocados en la cabeza del usuario hizo que, en su origen, las aplicaciones del BCI fueran únicamente médicas. En cambio, la nueva era tecnológica trajo consigo un auge del movimiento *Maker*, el *Do It Yourself* (DIY) y la búsqueda de nuevos usos para esta tecnología, entre otras muchas cosas. En este contexto, aparecen dispositivos BCI para uso no médico, lo que hace que las investigaciones en este ámbito se incrementen exponencialmente, hasta el punto de usarse para ocio y videojuegos.

Sin embargo, cabe destacar que la existencia de dispositivos comerciales a un precio relativamente asequible para investigadores y usuarios no desmerece a la tecnología subyacente, que es de una alta complejidad.

Más concretamente, una interfaz cerebro-ordenador se encarga de traducir las señales cerebrales en información útil para un *software*, que podrá realizar acciones sobre dispositivos externos.

Para ello, el sistema contiene una serie de sensores que miden las señales cerebrales, un amplificador para mejorar la calidad de las débiles señales obtenidas, y un ordenador que traduce estas señales en comandos para controlar algún programa o dispositivo.

Por tanto, el concepto básico sobre el que se desarrolla la tecnología BCI es la interpretación, en tiempo real, de la actividad de una población de neuronas y su traducción a una acción directa efectuada por un dispositivo externo. La Figura 2.1 muestra un esquema simplificado de una aplicación BCI [3].

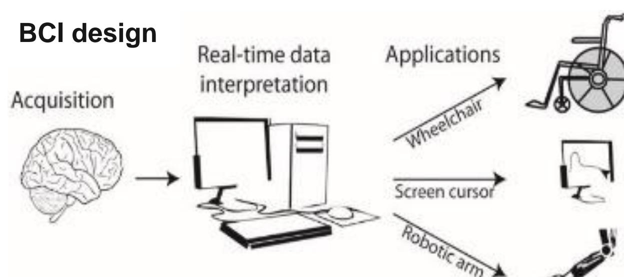


Figura 2.1 Representación esquemática de una aplicación BCI

El desarrollo de una interfaz cerebro-ordenador incluye dos fases bien diferenciadas:

1. Fase de entrenamiento: durante esta se aprende a asociar la actividad neuronal con el estado actual de las variables de interés: posición de un estímulo en el espacio, dirección de un móvil, localización espacial de algún punto de atención…

2. Fase de pruebas: durante esta se usan los resultados definidos en la fase de entrenamiento para asignar el estado más probable de la variable de interés. Para ello se analiza la actividad neuronal observada ante un estímulo de los aprendidos durante el entrenamiento.

Sin embargo, la conversión entre señales cerebrales y señales de control para actuadores puede realizarse mediante distintos métodos, que se resumen en diseños en lazo abierto y en lazo cerrado.

En la forma más simple, la cohesión entre el la actividad neuronal y la salida deseada depende de la interpretación de las señales cerebrales. Por ejemplo, identificar la actividad cerebral con la que se codifica un cierto movimiento permite asociar la actividad de un grupo de neuronas (como el movimiento de un brazo) con un actuador de salida externo y específico (como mover un cursor en una pantalla). En la Figura 2.2 se muestra un caso de diseño en lazo abierto en el que el sujeto elige un punto de la pantalla y se graba la actividad cerebral de su región cortical, sin realimentación.

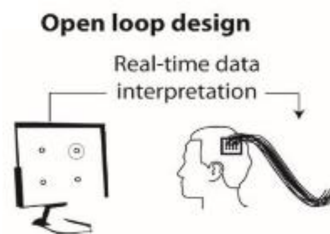


Figura 2.2 BCI en lazo abierto

Sin embargo, hay otros diseños más complejos basados en las capacidades de adaptación del córtex. En este caso, se dependerá del proceso de aprendizaje y los métodos de refuerzo positivo. Para ello, los sujetos aprenden a controlar el actuador mediante una realimentación sensorial, como ver la trayectoria que sigue el cursor, lo que les permite asegurarse de cómo están controlando la salida. Se trata de diseños en lazo cerrado, como el mostrado en la Figura 2.3, en el que el sujeto ve el cursor en la pantalla y en tiempo real tiene que adaptar su actividad cerebral para mejorar la precisión de la trayectoria.

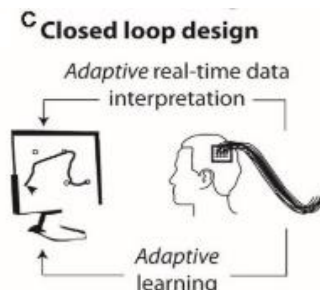


Figura 2.3 BCI en lazo cerrado

2.2 Fundamentos del BCI

Como se ha mostrado en la Figura 2.1, el desarrollo de una interfaz cerebro-ordenador sigue un proceso que consta básicamente de:

adquisición + interpretación + aplicación

El presente apartado se centra en la explicación del proceso de adquisición: de dónde se obtienen las señales y qué distintos métodos hay para obtenerlas.

2.2.1 Señales cerebrales

El BCI trata de recoger e interpretar o decodificar señales cerebrales, que son las que permiten la comunicación entre neuronas mediante el envío y recepción de dichas señales. Es posible acceder a estas señales, generalmente conocidas como actividad cerebral, mediante sensores avanzados.

De acuerdo con el entendimiento actual, el rol del sistema nervioso central (SNC) es responder a estímulos externos e internos mediante la generación de unas determinadas salidas. Dichas salidas, al hacer referencia al SNC, son tanto neuromusculares como hormonales. Mediante una interface cerebro-ordenador, se puede hacer uso de una nueva salida que reemplaza, restaura, aumenta, complementa o mejora las salidas naturales del SNC [4].

Para entender correctamente este concepto es necesario, en primer lugar, aclarar el término Sistema Nervioso Central. El SNC está compuesto del cerebro y la médula espinal, y es distinto del sistema nervioso periférico (SNP) que incluye los nervios periféricos, ganglios y receptores sensoriales. El SNC presenta algunas características únicas como son su localización en las meninges, su tipo de célula distintivo y su histología, así como su rol en la integración de las numerosas entradas sensoriales para producir salidas motoras eficaces [5].

Por tanto, la actividad del SNC comprende los fenómenos electrofisiológicos, neuroquímicos y metabólicos, que ocurren continuamente en dicho sistema. Esto

puede monitorizarse mediante la medición de campos eléctricos o magnéticos, la oxigenación de la hemoglobina u otros parámetros. Para ello se hace necesario el uso de elementos sensores y determinadas técnicas de mayor o menor complejidad.

2.2.2 Medición de las señales cerebrales

Las señales cerebrales adquiridas mediante métodos electrofisiológicos y hemodinámicos pueden usarse como entradas para un sistema BCI. Sin embargo, este tipo de señales difieren en gran medida en cuanto a resolución topográfica, contenido de frecuencia, área de origen y necesidades técnicas. En la Figura 2.4 [5] se muestran los principales métodos electrofisiológicos, para los cuales los electrodos pueden situarse en el cuero cabelludo, en la superficie del cerebro o en el interior del mismo.

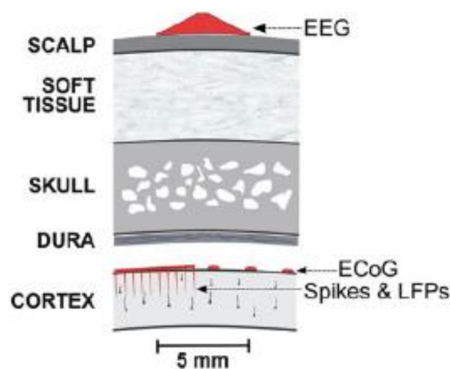


Figura 2.4 Zonas de colocación de sensores

Una de las técnicas más usadas para medir la actividad cerebral y realizar análisis neurológicos médicos es el electroencefalograma (EEG). Para llevarla a cabo se usan sensores que se distribuyen sobre el cuero cabelludo del paciente, y su resolución es del orden de un centímetro. Un ejemplo se muestra en la Figura 2.5.

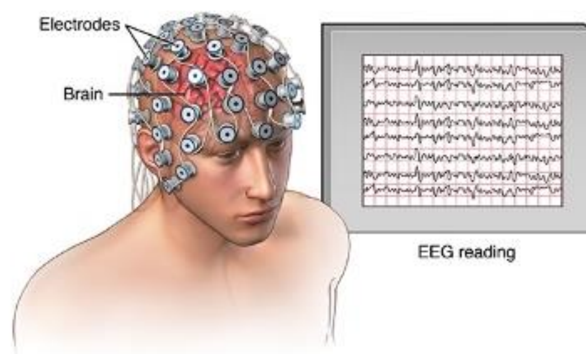


Figura 2.5 Electroencefalograma

Sin embargo, los electrodos también se pueden colocar directamente en o sobre el tejido cerebral, para lo que es necesario un procedimiento quirúrgico. Cuando se colocan sobre la superficie cortical recibe el nombre de electrocorticograma (ECoG), que no produce daños en el cerebro y además permite obtener unas señales de calidad significativamente superior a las obtenidas por técnicas no invasivas como el EEG. La resolución que presenta es del orden de un milímetro.

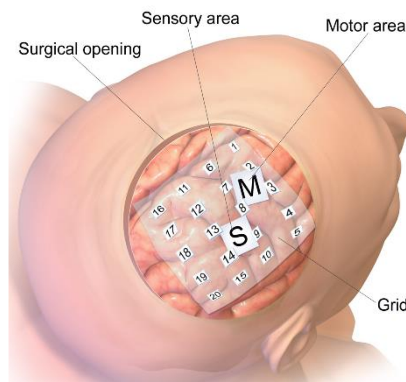


Figura 2.6 Electrocorticograma

En el caso de que se requiera medir directamente los potenciales de acción neuronales (*spikes*) o los potenciales de campo local (LFP, por sus siglas en ingles *Local Field Potentials*), será necesario insertar arrays de microelectrodos en determinadas regiones del cerebro. Un ejemplo de este tipo de array se muestra en la Figura 2.7.

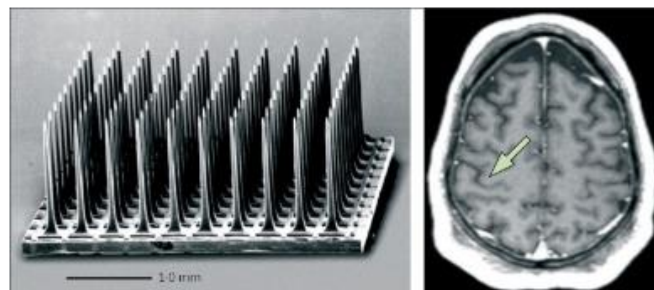


Figura 2.7 Array multielectrodo intracortical

Tal y como se ha mencionado previamente, existen otros métodos más allá de los electrofisiológicos: los hemodinámicos. Estos incluyen técnicas como la Imagen por Resonancia Magnética funcional (fMRI por sus siglas en inglés *functional Magnetic Resonance Imaging*), que mide la actividad cerebral con un escáner fMRI

como el de la Figura 2.8. Para ello se usan fuertes campos magnéticos y ondas radio para obtener una imagen del cerebro. Este método se basa en la detección de cambios asociados al flujo sanguíneo (concretamente el nivel de oxígeno sanguíneo), teniendo en cuenta el hecho de que éste y la activación neuronal están altamente relacionados: cuando un área del cerebro está en uso, se incremente el flujo sanguíneo en dicha región como se observa en la Figura 2.9.

Figura 2.8 Escáner fMRI de Siemens (MAGNETOM® Spectra)

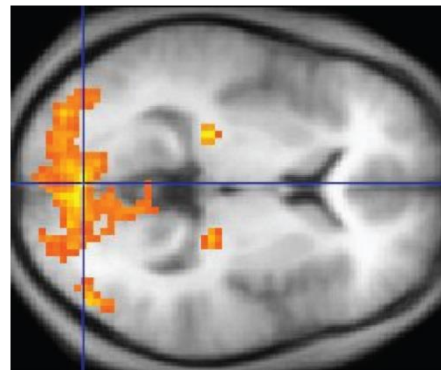


Figura 2.9 Imagen fMRI con áreas amarillas mostrando el incremento de la actividad cerebral comparada con la condición de control.

Aparte del fMRI, se mide la actividad cerebral mediante Magnetoencefalografía (MEG), usando magnetómetros sensibles como el de la Figura 2.10. Se trata de una técnica de neuroimagen para representar un mapa del cerebro, como el de la Figura 2.11, según la actividad de cada región analizando los campos magnéticos producidos por las corrientes eléctricas que se generan de forma natural en el cerebro.

Figura 2.10 Escáner MEG de Elekta (Neuromag® TRIUX)

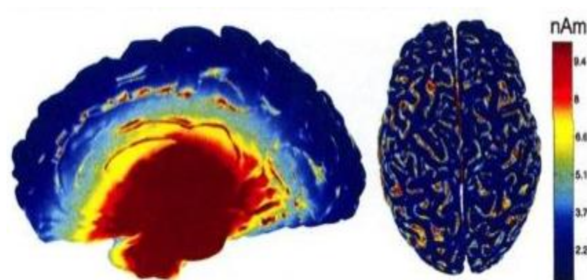


Figura 2.11 Imagen MEG mostrando la intensidad de la actividad cerebral

Aunque los resultados obtenidos con estas máquinas son de gran calidad y se trata de tecnologías maduras, estos procedimientos cuentan con ciertos inconvenientes que imposibilitan su uso para BCI. Como podemos observar en las Figura 2.8 y Figura 2.10, las máquinas de fMRI y MEG son de gran tamaño, además de tener un precio elevado (del orden del millón de dólares). Por tanto, su acceso está limitado a usos médicos o de investigaciones de alto nivel.

Existe una técnica adicional, llamada Espectroscopia del Infrarrojo Cercano (NIRS, por sus siglas en inglés *Near-Infrared Spectroscopy*), que mide la actividad cerebral mediante el uso de luz con longitud de onda cercana al infrarrojo propagándose a través del cráneo. Esta técnica puede realizarse de forma portátil sin necesidad de grandes máquinas, y además es muy segura y accesible, lo cual beneficiaría en gran medida su uso para BCI. Sin embargo, todavía es una tecnología incipiente y los resultados obtenidos mediante NIRS son inferiores a los obtenidos mediante EEG [6].

2.3 Evolución del BCI

El origen del BCI es el resultado de una serie de investigaciones sobre el cerebro humano llevados a cabo desde principios del siglo XX. Hans Berger, psiquiatra francés, es considerado como el inventor de la electroencefalografía (EEG) tras realizar un estudio sistemático de la actividad eléctrica del cerebro humano. Fue en 1924 cuando realizó la primera grabación de la actividad cerebral en un humano. Sin embargo, en 1875, Richard Caton, físico y psicólogo británico, ya había estudiado las ondas cerebrales de ciertos animales [7] [8].

Caton centró sus investigaciones en observar los impulsos eléctricos que existen en la superficie del cerebro de seres vivos, especialmente en conejos, perros y monos. Estos estudios sirvieron como una sólida base sobre la que Berger realizó numerosos descubrimientos. Fue el propio Berger quien citó a Caton en sus estudios, indicando que había sido este último quien ya había publicado experimentos realizados en el cerebro de perros y simios, en los cuales se habían situado electrodos

desnudos en la zona del córtex cerebral y en la superficie del cráneo. Concretaba también que había usado un galvanómetro sensible para medir las variaciones de corriente que tenían lugar en el cerebro de los animales, llegando a la conclusión de que dicha variación se hacía mayor durante el sueño y cuando se acerca el momento de la muerte, mientras que después de la misma, cuando cesa la actividad cerebral, comenzaba a debilitarse hasta desaparecer por completo.

Siguiendo los estudios de Caton, Berger comenzó a investigar con el cerebro humano. El hecho de que Berger fuese pionero en adquirir datos del cerebro humano le permitió ser el primero en analizar y describir los distintos ritmos que estaban presentes en cerebros normales y anormales. Como consecuencia, descubrió la existencia de las ondas alfa, a partir de entonces también conocidas como ondas de Berger, así como las beta y otras. Por otro lado, también realizó estudios acerca de la naturaleza de alteraciones que observaba en los EEG, que asoció a desórdenes cerebrales tales como la epilepsia.

El método que empleó para realizar el EEG fue la inserción de cables de plata bajo el cuero cabelludo, uno en la parte frontal del cráneo y otro en la parte occipital. Posteriormente, comenzó a usar electrodos construidos con finas láminas de plata que se sujetaban a la cabeza mediante un vendaje de tejido elástico. Respecto a los dispositivos de adquisición y grabación de los datos, comenzó usando el electrómetro de capilaridad de Lippmann, que se muestra en la Figura 2.12 [9].

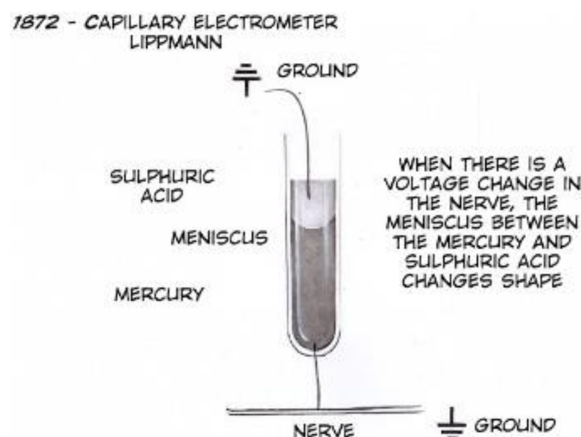


Figura 2.12 Electrómetro de capilaridad de Lippmann

Éste consistía en un tubo de ensayo lleno de mercurio y cubierto con una capa de ácido sulfúrico conectado mediante un cable a un nervio humano. Cuando la tensión del nervio cambiaba debido a un potencial de acción, la forma del menisco entre el mercurio y el ácido sulfúrico cambia, lo cual es observable bajo microscopio. Este amplificador en estado líquido era muy sensible, pero tenía serios inconvenientes como el hecho de que el menisco no cambiaba lo suficientemente rápido como para aislar un potencial de acción de forma individual. Cuando Berger empleó este método como forma de adquirir señales EEG, encontró dificultades que le llevaron a no obtener los resultados deseados.

Consecuentemente, cambió esta técnica por el galvanómetro de Einhoven, que se muestra en la Figura 2.13, el más sensible inventado hasta el momento. Consistía en un hilo de vidrio bañado en oro que se encontraba suspendido entre dos electroimanes de alta potencia, de forma que el hilo oscilaba cuando un impulso eléctrico atravesaba el nervio. Para registrar el movimiento se hacía uso de una luz y se grababa la sombra en un papel fotográfico. Cabe destacar que fue el mismo Einhoven quien hizo uso de su propio invento para grabar señales de electrocardiograma [9].

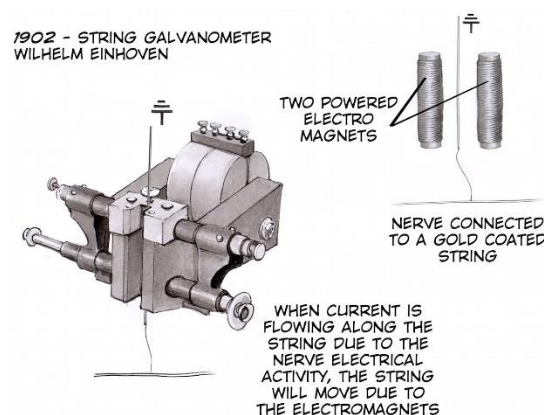


Figura 2.13 Galvanómetro de Einhoven

Más adelante, se decantó por usar el galvanómetro de doble bobina de Siemens, que le permitió obtener señales eléctricas del orden de los 0.1 mV. El resultado obtenido, que tenía una duración máxima de 3 segundos, era fotografiado

por un asistente. Un ejemplo de una de estas fotografías se muestra en la Figura 2.14.

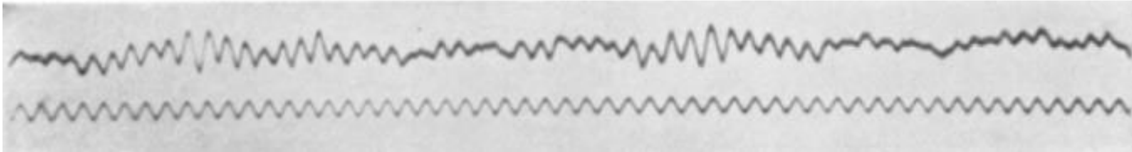


Figura 2.14 Grabación EEG realizada por Berger

Las mejoras en las técnicas de electroencefalograma permitieron que, en los años 60, Grey Walter usara una señal EEG obtenida de forma no invasiva para controlar un proyector [10] y Eberhard Fetz enseñara a unos monos a controlar un medidor de aguja cambiando la tasa de disparo de una única neurona cortical [11] [12]. En la Figura 2.15 se muestra un esquema del proceso final al que llegó tras años de investigación.

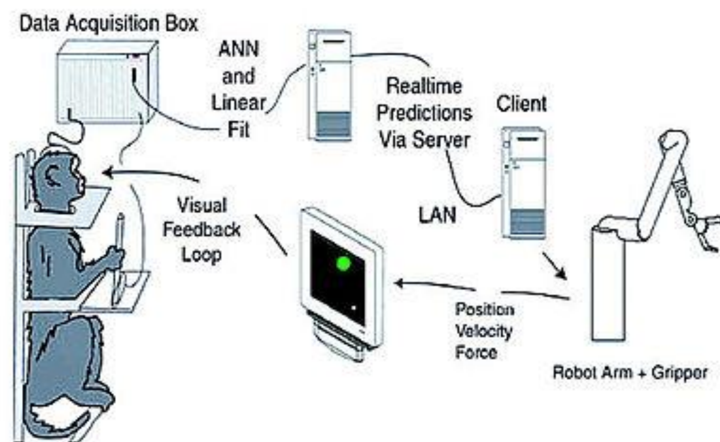


Figura 2.15 Esquema del experimento de Fetz con monos

En los años 70 comenzaron unas nuevas investigaciones sobre BCI en la Universidad de California, Los Angeles, gracias a una beca recibida de la Fundación Nacional para la Ciencia (NSF, por sus siglas en inglés *National Science Foundation*), seguida de un contrato por parte de la Agencia de Investigación de Proyectos Avanzados de Defensa (DARPA, por sus siglas en inglés *Defense Advanced Research Projects Agency*). Jacques Vidal desarrolló un sistema que usaba potenciales visuales evocados obtenidos de forma no invasiva para determinar la dirección de mirada en humanos. Con ello, consiguió controlar un cursor de

ordenador, moviéndolo por la pantalla con su mirada. Como resultado de estos estudios, se publicaron una serie de artículos que supusieron la primera aparición del término interfaz cerebro-ordenador en la literatura científica [13] [14].

Unos años después, en 1980, Elbert et al. mostraron cómo la gente podía aprender a controlar los potenciales corticales lentos obtenidos mediante EEG y usar dicho control para mover una imagen en una pantalla de televisión [15].

En el año 1988, se publicó el artículo "*Using EEG alpha rhythm to control a mobile robot*" (Uso de ondas alfa EEG para controlar un robot móvil), que fue la primera aplicación documentada sobre el control de un objeto físico usando señales EEG [16].

Las investigaciones en el campo del BCI comenzaron a crecer rápidamente a partir de mediados de los 90, y este crecimiento ha continuado hasta el presente. Cabe destacar el trabajo realizado por Chapin et al. en 1999, que demostró que la actividad cerebral del córtex motor de una rata podía usarse para controlar un brazo robótico.

El trabajo realizado en los últimos 25 años ha incluido un amplio rango de estudios en todas las áreas relativas al BCI, incluyendo neurociencia básica, aplicaciones, ingeniería biomédica, materiales, señales eléctricas, procesamiento de señales, ciencia computacional, tecnología de asistencia, rehabilitación clínica y otros factores [17]. Todos estos logros han supuesto grandes avances en la tecnología BCI y han sacado a la luz el gran potencial de esta.

2.4 Curvas de expectación de Gartner

2.4.1 Introducción

Las curvas de expectación de Gartner, conocidas a nivel internacional como *Gartner Hype Cycle*, son una representación gráfica de la madurez de ciertas tecnologías y aplicaciones, y cómo éstas son potencialmente relevantes para resolver

problemas actuales y dar lugar a nuevas oportunidades. Por tanto, se puede afirmar que es una forma rápida de ver cómo va a evolucionar en el tiempo cierta tecnología, con el objetivo de poder valorar una posible inversión en alguna de ellas [18].

En el ciclo de vida de determinada tecnología en desarrollo se pueden diferenciar cinco fases clave, que son las representadas en la Figura 2.16:

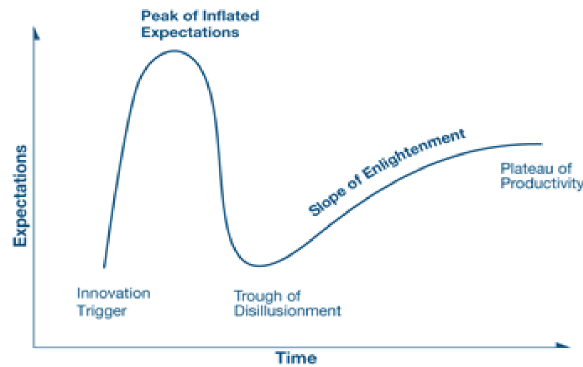


Figura 2.16 Fases del ciclo de vida de una tecnología

- **Lanzamiento** (*Innovation Trigger*): aparece una tecnología que se cree potencialmente útil para algún aspecto concreto. Cada prueba de concepto conlleva publicidad para la sociedad e incrementa el interés mediático. Normalmente, en esta etapa no existen productos aptos para el uso ni se ha probado la viabilidad comercial de la tecnología.

- **Pico de expectativas sobredimensionadas** (*Peak of Inflated Expectations*): la publicidad inicial que se ha dado a la tecnología da lugar a un gran número de investigaciones, algunas de ellas exitosas, y otras que terminan en fallos. Ciertas empresas comienzan a tomar parte en el desarrollo de esta tecnología e invierten en materiales e I+D+i, mientras que otras prefieren mantenerse al margen ante la posibilidad de que la tecnología fracase.

- **Abismo de desilusión** (*Trough of disillusionment*): el interés en la tecnología se debilita debido a fallos en los experimentos y en la implementación. Los productores deben empezar a realizar cambios

significativos o fracasarán. La inversión continua solo si los proveedores supervivientes mejoran sus productos hasta conseguir la satisfacción del mercado.

- **Rampa de consolidación** (*Slope of enlightenment*): Aparecen nuevas formas en que la tecnología en desarrollo podría beneficiar a la empresa, lo que conlleva que se expanda su entendimiento y comience a arraigar. Se desarrollan productos de segunda y tercera generación y más empresas invierten en ellos, aunque las compañías más conservadoras permanecen cautas.

- **Meseta de productividad** (*Plateau of productivity*): la adopción generalizada comienza a ser una realidad. Se definen con mayor claridad los criterios para asegurar la viabilidad. El alto grado de aplicación de la tecnología en el mercado y la relevancia que ha adquirido son la muestra del éxito alcanzado.

A continuación, se muestra en la Figura 2.17 una imagen que resume todos estos puntos, explicando la situación en que se encuentra la tecnología respecto al mercado según su situación en la curva.

Figura 2.17 Esquema de las zonas de las curvas de expectación de Gartner

2.4.2 Aplicación al BCI

Se considera interesante analizar la evolución de la tecnología BCI desde el punto de vista de las curvas de Gartner. Para ello, se van a mostrar las gráficas de los últimos años (2013-2016). Esto es debido a que en la curva de 2012 no aparece como tecnología emergente el BCI, sino que se considera que es a partir de 2013 cuando comienza su auge y empieza a aumentar su expectación [19].

La Figura 2.18 muestra la situación de BCI en la curva de 2013. Se observa que el marcador usado para cada tecnología es distinto, y sus significados se muestran en la leyenda de la imagen: hacen referencia al tiempo que habría de pasar para que una tecnología llegara a alcanzar la llamada “meseta de productividad”, es decir, para alcanzar la consolidación en el mercado [20].

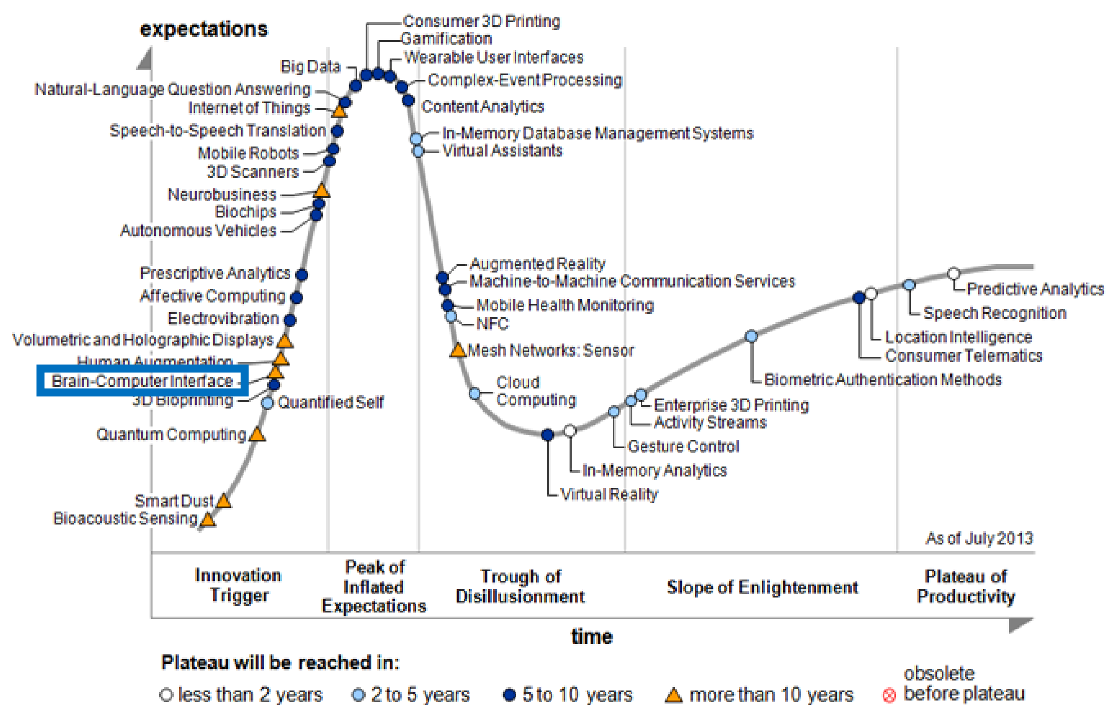


Figura 2.18 Curva de Gartner 2013

En la Figura 2.18 se ha marcado la situación del BCI en 2013. Como se puede observar, el icono asignado es el triángulo amarillo, que indica que se alcanzará la estabilidad en más de 10 años.

A continuación, en la Figura 2.19, se observa la curva de 2014. La situación del BCI en ella es prácticamente idéntica a la del año previo. Se asume, por tanto, que ese año no supuso ningún cambio significativo en la expectación de esta tecnología, que permaneció estancada [21].

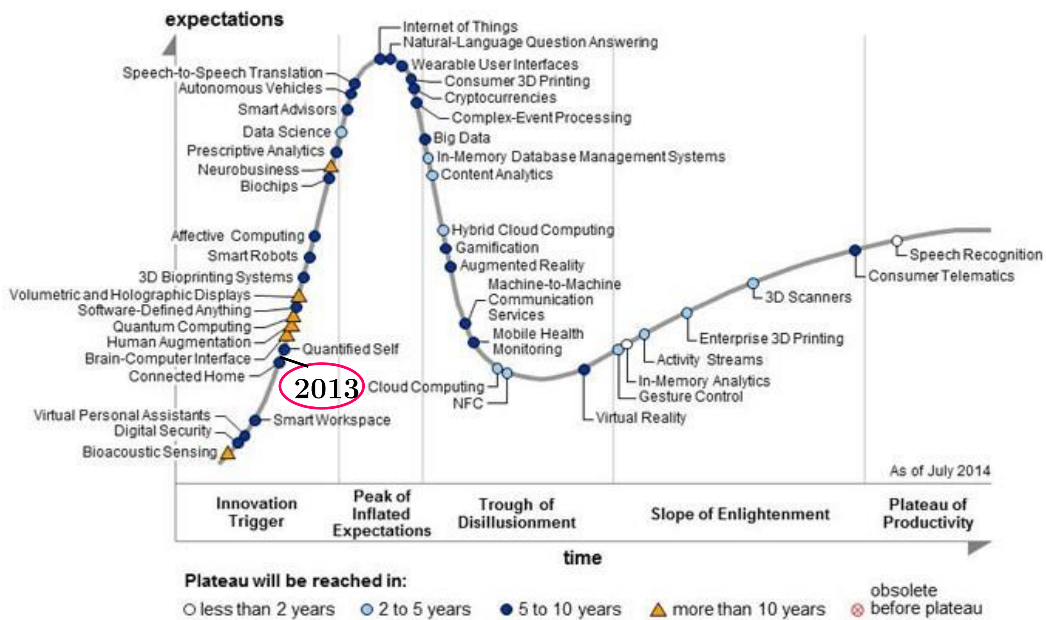


Figura 2.19 Curva de Gartner 2014

Siguiendo con la misma dinámica, se muestra la curva de 2015 en la Figura 2.20. La evolución es mayor que la experimentada entre 2013 y 2014 [22].

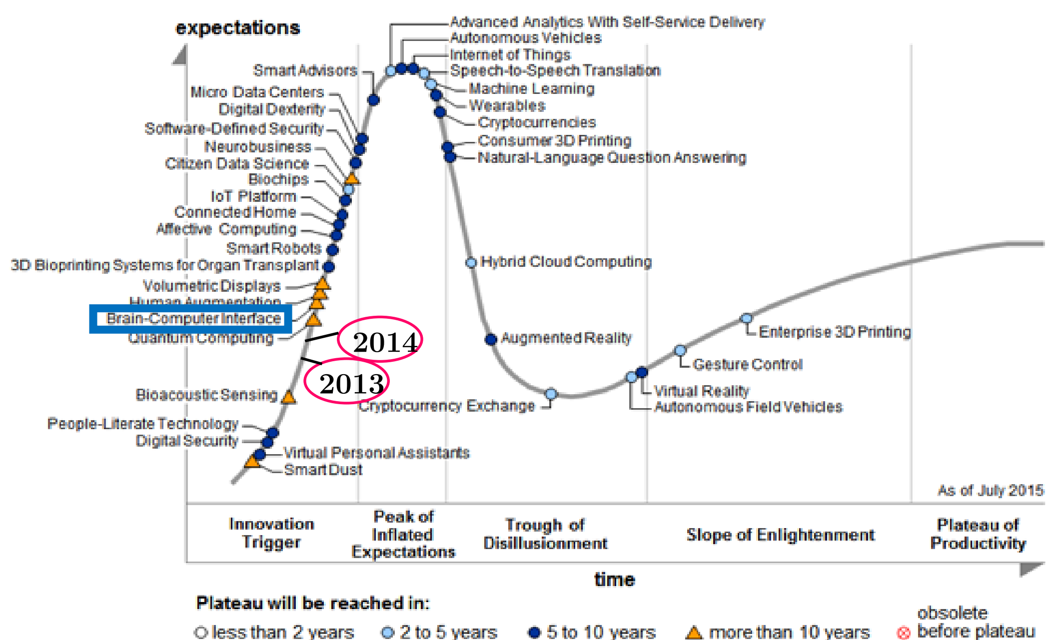


Figura 2.20 Curva de Gartner 2015

Por último, se muestra en la Figura 2.21 la curva de 2016. En dicha imagen se observa que el crecimiento entre un año y el anterior se ha ido incrementando conforme se acerca a la actualidad [23].

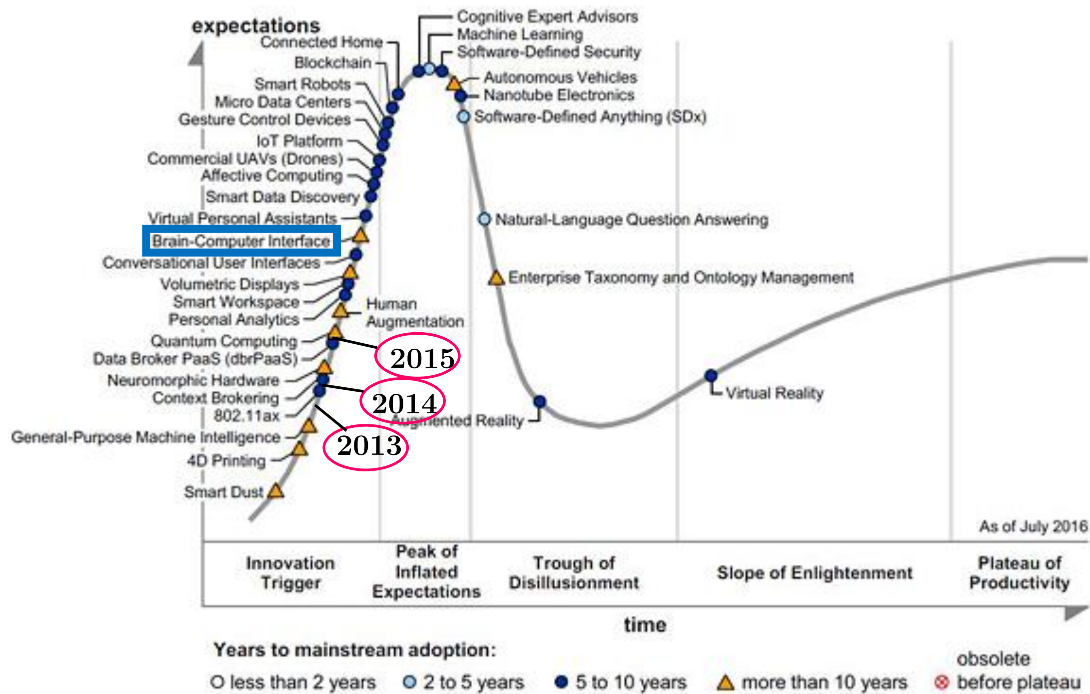


Figura 2.21 Curva de Gartner 2016

Se concluye citando el hecho de que la tecnología está creciendo, la expectación hacia la misma está aumentando a la vez que se van consolidando numerosos avances. Según la Figura 2.17, a día de hoy el BCI estaría entre las fases de primera generación de productos e investigación, por lo que se hace necesario la creación de nuevas líneas de investigación con el objetivo final de que llegue a ser una tecnología madura, sin caer en el olvido durante el Abismo de la Desilusión.

2.5 *Hardware* disponible

2.5.1 BCI

Actualmente, en el mercado existen diversos modelos de casco y elementos sensores específicos para la tecnología BCI. A continuación, se hará una breve

exposición de los que se ofrecen al público. También se indicarán los que fueron considerados como opción para la realización del presente proyecto.

2.5.1.1 Emotiv Insight

El Emotiv Insight (Figura 2.22) es un casco de 5 canales, inalámbrico con batería de 480 mAh que graba ondas cerebrales mediante EEG y las traduce a información fácilmente entendible y aplicable por el usuario. Los sensores son secos, van incluidos con el casco y son de un polímero de gran conductividad eléctrica que absorbe la humedad del ambiente. Gracias a esto, se suprime la necesidad de usar geles o soluciones salinas para mejorar el contacto del sensor [24].

Figura 2.22 Emotiv Insight

En su precio, que es de 299$, incluye aplicaciones gratuitas como un panel de control y un simulador, además del Kit de Desarrollo *Software* (SDK, por sus siglas en inglés *Software Development Kit*). Sin embargo, para acceder a los datos de las señales se debe adquirir una licencia aparte.

2.5.1.2 Emotiv Epoc+

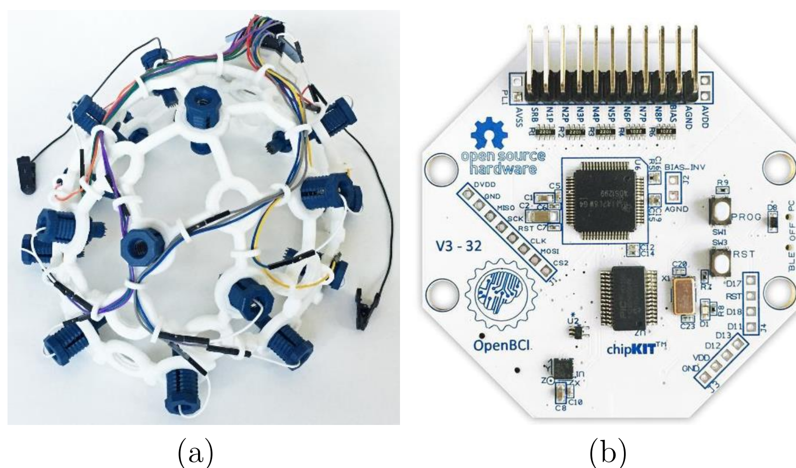
El Emotiv Epoc+ (Figura 2.23) es un casco de 14 canales, inalámbrico, diseñado para la investigación y el desarrollo de aplicaciones BCI avanzadas. Respecto a los sensores, en este caso no son secos, por lo que es necesario mojarlos con un gel salino que dificulta y alarga las puestas en marcha. Otra diferencia en relación al Emotiv Insight es la batería: en este caso es de 680mAh, lo que le otorga más de 6 horas de autonomía usando Bluetooth de Baja Energía (BTLE, por sus siglas en inglés *BlueTooth Low Energy*).

Figura 2.23 Emotiv EPOC+

Se trata de un sistema de alta resolución cuyo precio es de 799$. Al igual que el Emotiv Insight, incluye las aplicaciones gratuitas y el SDK de Emotiv, pero no la licencia para poder tratar las señales.

2.5.1.3 OpenBCI

OpenBCI (Figura 2.24) es una plataforma *open source* para BCI. Los dispositivos que ofertan se puedan usar para medir la actividad eléctrica cerebral (EEG), muscular (EMG) y cardíaca (EKG), siendo compatible con electrodos EEG estándar. Sin embargo, no está preparado para ser usado con electrodos secos tal y como mencionaba uno de los creadores, Conor Russomanno: “Elegimos electrodos húmedos porque nadie ha creado un electrodo seco efectivo que pueda obtener una señal de calidad a través del pelo” [25].



(a) (b)

Figura 2.24 OpenBCI. (a)Casco Ultracortex "Mark IV" EEG (b)Cyton Biosensing de 8 canales

En es[...]ctura del casco (impresión 3D), los electrodos y la placa, así como el gel conductivo. Se ofrecen

distintas posibilidades de compra variando el número de canales de la placa y cómo se quiera recibir el casco (sin montar, montado o solo los archivos para imprimirlo uno mismo) [26].

Por ejemplo, en caso de disponer de impresora 3D y adquirir la placa de 8 canales, el precio total sería de 850$, a lo que habría que sumar el resto de material necesario.

2.5.1.4 Neurosky MindWave Mobile EEG

El casco MindWave Mobile EEG, de Neurosky, es un dispositivo muy sencillo de poner debido a su sencilla estructura, mostrada en la Figura 2.25. Está diseñado con el objetivo de ser usado para videojuegos e incluso para técnicas de relajación, para lo que ofrece unas determinadas aplicaciones. Por otro lado, ofrecen herramientas de investigación que incluyen interfaces para visualizar señales y analizarlas [27].

Figura 2.25 Neurosky MindWave Mobile EEG

Se puede adquirir un kit de desarrollo que incluye el casco y tres aplicaciones para visualización y tutoriales que tiene un precio de 100$. Sin embargo, si se quiere adquirir el paquete de investigación habrá que hacerlo aparte por el precio de 500$ [28].

2.5.1.5 BrainProducts

BrainProducts ofrece una amplia gama de productos relacionados con la adquisición de señales cerebrales, potenciales evocados... mediante distintas técnicas. Se trata de dispositivos médicos, de gran calidad, diseñados incluso para trabajar con grandes máquinas, por ejemplo, de fMRI (Figura 2.8). A continuación, en la Figura 2.26, se muestra un sistema EEG de electrodo seco con malla.

Figura 2.26 BrainProducts actiCAP Xpress

La adquisición de este tipo de productos de alta calidad ha de efectuarse por medio de distribuidores, por lo que el precio no está disponible en su página web.

2.5.1.6 Olimex EEG-SMT

OpenEEG es un proyecto de Olimex que se inició con el objetivo de crear dispositivos EEG comerciales a un precio asequible para gente amateur que está interesada en el campo del BCI pero no va a realizar investigaciones de gran envergadura. Disponen de varias placas y de un dispositivo con carcasa, 2 canales y comunicación USB [29]. Este se muestra en la Figura 2.27.

Respecto a los electrodos, serán 5 pasivos o 4 activos y uno pasivo, que podrán conectarse directamente en la carcasa.

Figura 2.27 Olimex EEG-SMT

El precio del EEG-SMT es de 99$ e incluye también el acceso completo a OpenEEG, donde se pueden encontrar distintas aplicaciones y ejemplos.

2.5.1.7 Muse

Muse (Figura 2.28), de Interaxon, es un dispositivo que adquiere señales cerebrales con el objetivo de ayudarte a saber tu estado actual y relajarte mediante meditación. Sin embargo, ofrece también un SDK para desarrollar aplicaciones Android e iOS gracias a un conjunto de ejemplos y librerías [30].

Los sensores que utiliza son secos, lo que permite una fácil y rápida conexión. Desde la aplicación se puede ver la calidad de la conexión y se envía información sobre el estado del usuario.

Figura 2.28 InterAxon Muse

Su precio es de 289$ e incluye unos auriculares que conectar al dispositivo móvil para la meditación.

Con el objetivo de elegir el dispositivo BCI que se usaría en el presente TFG y que pasaría a formar parte del departamento de Ingeniería de Sistemas y Automática UPCT, se valoraron las opciones arriba planteadas. Considerando la complejidad de las aplicaciones que se desarrollan, el precio, el hecho de que en el departamento se había trabajado previamente con Emotiv y, muy importante, el que los electrodos fuesen secos para mayor comodidad, se concluyó que se adquiriría el Emotiv Insight (Figura 2.22).

2.5.2 Aplicación robótica

El auge que está experimento el BCI a lo largo de los últimos años es debido, principalmente, a la cantidad de aplicaciones que se están desarrollando. Algunas de ellas se tratan en el control de un sistema robótico como un dron, un robot tipo coche, un brazo o mano robótica... A continuación, se muestran tres ejemplos de aplicaciones del BCI.

2.5.2.1 Dron

En 2016, Florida fue sede de la primera carrera de drones controlados únicamente con un casco BCI, en este caso el Emotiv Insight (Figura 2.29). Lo que se busca con este tipo de eventos es popularizar el uso del BCI, animando a la gente a hacer aplicaciones y mostrarlas [31].

Figura 2.29 Carrera de drones BCI

2.5.2.2 Robot Lego

En este caso, el objetivo del BCI es el control de un robot móvil Lego Mindstorms NXT usando la aplicación de Emotiv (Figura 2.30). Para ello se hace uso de los comandos mentales que el panel de control ofrece como son: tirar, empujar, rotar a la izquierda y rotar a la derecha [32].

Figura 2.30 Control de LEGO Mindstorms NXT con BCI

2.5.2.3 Mano robótica

Uno de los proyectos desarrollados por el co-fundador de OpenBCI fue el control mental de una mano robótica (Figura 2.31). Para ello utilizó el casco Ultracortex. Este experimento formó parte del *NYC media Lab* y permitió dar a conocer la tecnología de OpenBCI [33].

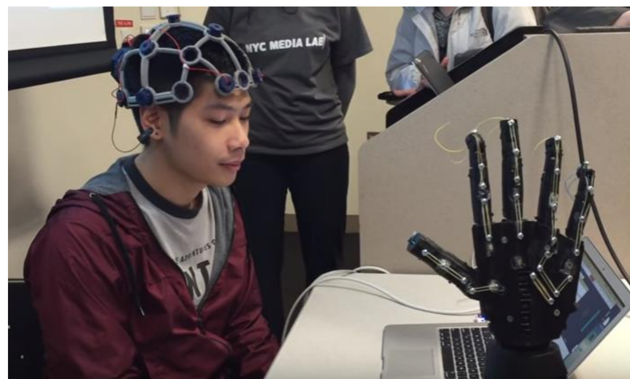


Figura 2.31 Control de mano robótica con BCI

Actualmente, se está descubriendo el potencial del BCI hasta tal punto que el control de una extremidad, en este caso una mano robótica, no se limita simplemente al movimiento articular, sino que también puede incorporar otros aspectos como la sensación de tacto [34].

Los ejemplos mostrados previamente (dron, coche lego y mano robótica) son tres de las múltiples aplicaciones desarrolladas para BCI, entre las que también se incluyen sistemas de escritura en ordenador, de movimiento de cursores…

Para el presente TFG la elección de la aplicación estuvo condicionada por la disponibilidad de elementos en el departamento. Puesto que se pretendía realizar un sistema robótico, la decisión estuvo entre usar la mano robótica Shadow Dexterous Hand (Figura 2.32) o el brazo robótico Kinova Mico (Figura 2.33).

Figura 2.32 Shadow Dexterous Hand

Figura 2.33 Kinova Mico[2]

En primer lugar, se comenzó a utilizar la Shadow Hand, pero las incompatibilidades con los sistemas operativos actuales y con las últimas versiones de ROS imposibilitaron la tarea. Por tanto, se decidió utilizar el brazo robótico Mico, de Kinova, que al ser más actual permitía la comunicación con las plataformas necesarias para el correcto funcionamiento del conjunto.

2.6 *Software* disponible

El *software* a utilizar en el proyecto es dependiente en gran medida del *hardware* elegido para el mismo. Tal y como se ha concluido en el apartado anterior, el dispositivo BCI que se ha usado es el Emotiv Insight, y la aplicación robótica hará uso del brazo Kinova Mico[2].

En primer lugar, Emotiv dispone de ciertas aplicaciones propias y un SDK, todo ello realmente útil para trabajar con BCI. Respecto al Kinova, incluye también un SDK muy completo para programar controladores para el brazo robótico.

El problema podría surgir al comunicar el casco con el brazo robótico, e incluso al integrar el control de sensores y actuadores del Kinova. Se requeriría de la programación independiente del Emotiv y del brazo y, posteriormente, de un *socket* para comunicar ambos. Sin embargo, la posibilidad de usar un *middleware* robótico facilitaría la tarea, puesto que permitiría la integración de los distintos elementos.

2.6.1 Orca

Orca es un framework de código abierto para el desarrollo de sistemas robóticos basados en sus componentes. Surge en 2003 a raíz de OROCOS, que nació en el año 2000 con el objetivo de ser un *software* de control robótico libre para investigación [35].

El propósito de Orca es proporcionar medios para definir y desarrollar los bloques que, integrados adecuadamente, forman sistemas robóticos complejos. Se basa en la reutilización del *software* mediante la definición de un conjunto de interfaces de uso común, proporcionando librerías con una API de alto nivel.

2.6.2 YARP

YARP (*Yet Another Robot Platform*) surge en 2002 como un paquete *software* de código abierto para interconectar los distintos elementos de un robot [36].

Incluye una serie de bibliotecas, protocolos y herramientas para mantener los módulos y dispositivos desacoplados. El objetivo principal es aumentar la duración de proyectos *software* de robots, que normalmente quedan obsoletos al actualizarse cualquier versión de algún componente.

A diferencia de Orca, cuya última revisión es de 2009, YARP se mantiene actualizada con versiones de 2017.

2.6.3 MIRA

MIRA (*Middleware for Robotic Applications*) surgió en 2012 como un middleware que proporciona diversas funcionalidades y numerosas herramientas para el desarrollo y pruebas de módulos *software*. Se centra también en la fácil creación de aplicaciones complejas y dinámicas a la vez que se reutilizan dichos módulos [37].

Actualmente, dispone de versiones recientes (2017) y se está usando para numerosas investigaciones en el campo de la robótica.

2.6.4 ROS

ROS es un middleware robótico que se desarrolló en 2007 y continúa siendo el más extendido para el desarrollo de aplicaciones robóticas. Además, es el eje en torno al cual se integra el presente proyecto [38].

Al igual que el resto de sistemas mencionados previamente, ROS no es un sistema operativo. Sin embargo, proporciona una capa de abstracción del hardware, permite el control de dispositivos en bajo nivel, el envío de mensajes entre procesos... para facilitar el desarrollo de aplicaciones y la comunicación entre dispositivos.

Capítulo 3

Emotiv Insight

3.1 Introducción

La finalidad del presente capítulo es mostrar una descripción detallada del Emotiv Insight, dispositivo BCI empleado en el TFG.

Como se anticipó en el Capítulo 1, el proyecto consta de una serie de elementos *hardware* y *software* que se encuentran convenientemente integrados. Por tanto, para describir adecuadamente el Emotiv se debe hacer referencia tanto a sus características físicas como al *software* con el que se ha utilizado en el proyecto.

3.2 Adquisición del dispositivo

Tras la elección del Emotiv Insight como dispositivo BCI a usar en el proyecto, se procedió a la adquisición del mismo.

La compra la efectuó el Departamento de Ingeniería de Sistemas y Automática por medio de la página web de Emotiv, desde el apartado *Hardware* (Figura 3.1).

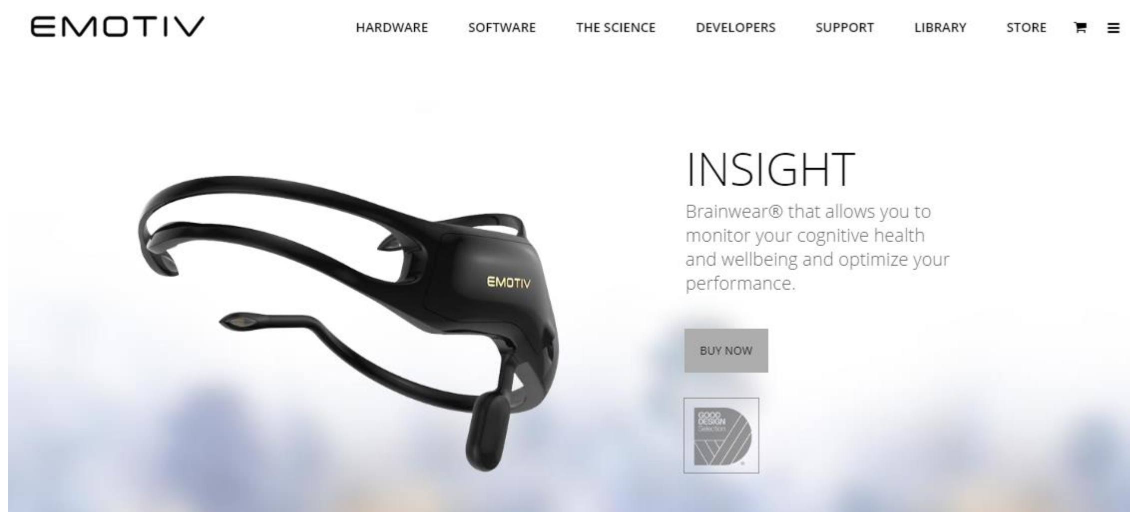


Figura 3.1 Página Web de Emotiv Insight

Además, al comprar el casco, se incluía:

- Los sensores necesarios para la adquisición de datos
- El cable de carga

Sin embargo, no se incluía el receptor universal USB que se necesita para los dispositivos que no disponen de protocolo Bluetooth SMART. Por tanto, éste se adquirió también para poder utilizar el Emotiv Insight con ordenadores con sistemas operativos Windows y Linux.

Por último, también se adquirió otro pack de sensores que servirían de repuesto, puesto que se recomienda cambiarlos cada 6 meses aproximadamente.

3.3 Descripción del *hardware*

A continuación, se muestran de forma desglosada las características técnicas y demás especificaciones del dispositivo Emotiv Insight que se ha adquirido.

Para ello, se incluyen fotografías tomadas a los distintos elementos que se encontraban en la caja recibida. Además, se pueden encontrar también una serie de tablas que incluyen parámetros de funcionamiento y otra información relevante para el uso de Emotiv.

- Estructura del casco, que se separa en dos piezas como se observa en la Figura 3.2. El ensamblaje es simple, basta con introducir el cabezal marcado con una M en la ranura que se encuentra junto a la entrada del cargador.

Figura 3.2 Ensamblaje de Emotiv Insight

- Dos packs de sensores, el que se incluía al comprar el casco y otro de recambio. Cada uno de ellos contiene cuatro sensores, que se colocan en los extremos del casco. Además, incluye dos elementos para cubrir los sensores de referencia y dos para el que se encuentra en el cuerpo del casco. También incluye una llave Allen que servirá para acceder a todos los sensores que no se encuentran en los extremos. El contenido se observa en la Figura 3.3.

Figura 3.3 Contenido de un pack de sensores

- Un cable para cargar el dispositivo. Se trata de un Jack-USB que permite conectar y cargar el Emotiv Insight a cualquier puerto USB, tanto de un ordenador como de un adaptador a la red (Figura 3.4).

Figura 3.4 Cargador del Emotiv Insight

- Un receptor universal USB para bluetooth SMART que se observa en la Figura 3.5.

Figura 3.5 Receptor Universal USB para Bluetooth SMART

3.3.1 Especificaciones técnicas

El Emotiv Insight es un casco EEG portátil de 5 canales que adquiere las ondas cerebrales y las traduce a datos que se pueden comprender y utilizar para controlar otros dispositivos. Está diseñado específicamente para ser usado en el campo de la investigación del BCI, permitiendo obtener señales de forma robusta.

Concretamente, ofrece 5 sensores EEG y 2 de referencia, lo que le otorga una resolución espacial suficiente como para adquirir información de la actividad cerebral.

La Tabla 3.1 muestra las especificaciones técnicas del dispositivo.

Señales	5 canales: AF3, AF4, T7, T8, Pz
	2 referencias en configuración para cancelación de ruido CMS/DRL
Tecnología del sensor EEG	Polímero semi-seco de larga duración
Sensores de movimiento	9 ejes (3 de giro, 3 acelerómetros y 3 magnetómetros)
Resolución de las señales	Ratio de transmisión de datos: 128 muestras por segundo por canal
	Mínima resolución de tensión: 0.51μV least significant bit (LSB)
	14 bits por canal
Frecuencia de respuesta y comunicación	1-43 Hz
	Wireless: Bluetooh 4.0 LE
	Proprietary Wireless: 2.4GHz
Alimentación	Batería interna de polímero de litio 480mAh
	Más de 4 horas de funcionamiento
Plataformas soportadas	Windows Vista, 7, 8, 10 Linux (Ubuntu, Fedora) Mac OS X iOS 5+ Android 4.4.3+ (excepto Android 5.0)

Tabla 3.1 Especificaciones técnicas de Emotiv Insight

3.3.2 Puesta en marcha

Una vez se tienen todos los elementos necesarios, se procede a realizar la puesta en marcha del Emotiv Insight. Para ello, se ha de seguir una serie de pasos que se explican a continuación:

1. Cargar el dispositivo usando el cable de carga USB. Esto tardará entre 30 minutos y 2 horas, según la fuente de carga que se emplee y teniendo en cuenta que algunos ordenadores limitan la corriente en ciertos puertos USB (se recomienda usar el puerto de carga rápida en caso de que se tenga). Mientras carga, el indicador LED que se encuentra junto al botón de encendido del Insight se mostrará de color anaranjado, mientras que cuando esté cargado pasará a verde.

Nota: no se puede usar el dispositivo mientras se carga. Esto es debido a que la compañía no puede garantizar que la fuente de carga de batería sea segura puesto que cada usuario utilizará un método y lugar distinto. Como se trata de un dispositivo en contacto directo con la cabeza del usuario, sería necesario un alto grado de aislamiento para los circuitos internos, lo cual no es práctico por varias razones: incrementaría la complejidad y precio del dispositivo, así como su peso y tamaño. Por tanto, la solución empleada ante la incertidumbre respecto a la fuente de carga es deshabilitar el dispositivo mientras se encuentre conectado [39].

2. Una vez cargado, se conecta el receptor universal USB al ordenador y se realiza el emparejamiento con el dispositivo al encenderlo. Para ello es conveniente abrir la aplicación, puesto que en ella se muestra la calidad de la conexión y el significado de los indicadores luminosos (Figura 3.6) que presenta el receptor USB.

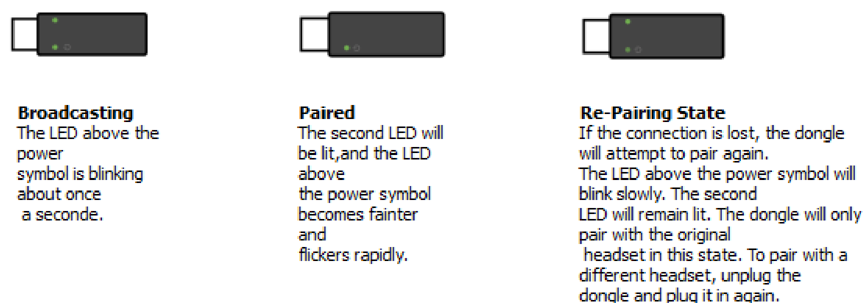


Figura 3.6 Significado del indicador luminoso del receptor USB

Tal y como se observa en la imagen anterior, el receptor USB tiene dos indicadores LED que informan sobre el estado de la conexión. Uno de ellos se encuentra sobre un símbolo de *power* tal y como se señala en la Figura 3.7.

Figura 3.7 Indicadores luminosos del receptor USB

- **Broadcasting**: el LED de power parpadea con un periodo de 1 segundo mientras que el otro está apagado.
- **Emparejado**: el LED de power pierde intensidad, se queda encendido muy leve y parpadea rápidamente. Esto es lo que se observa en la imagen de arriba.
- **Re-emparejando**: cuando se pierde la conexión, el receptor USB tratará de establecerla de nuevo. Se mostrará este proceso mediante el parpadeo lento del LED de power, mientras que el otro permanece encendido. Cabe destacar que, en este estado, el LED solo se emparejará con el casco al que estaba conectado antes de perder la conexión. Si se quiere emparejar con otro, se habrá de desconectar y volver a conectar el USB.

El emparejamiento no tarda más de unos segundos si todo funciona como es debido y se tiene el casco cerca del ordenador al que se conecte el receptor USB. Una vez emparejado, se puede comenzar a utilizar el Emotiv Insight.

Para empezar, es altamente recomendable utilizar la aplicación gratuita Emotiv Xavier Control Panel, que se puede obtener desde la página web de Emotiv. Una vez descargada, se instala fácilmente siguiendo las instrucciones que muestran por pantalla. Se recomienda también registrarse la primera vez e iniciar sesión cada vez que se acceda desde la pantalla principal, mostrada en la Figura 3.8.

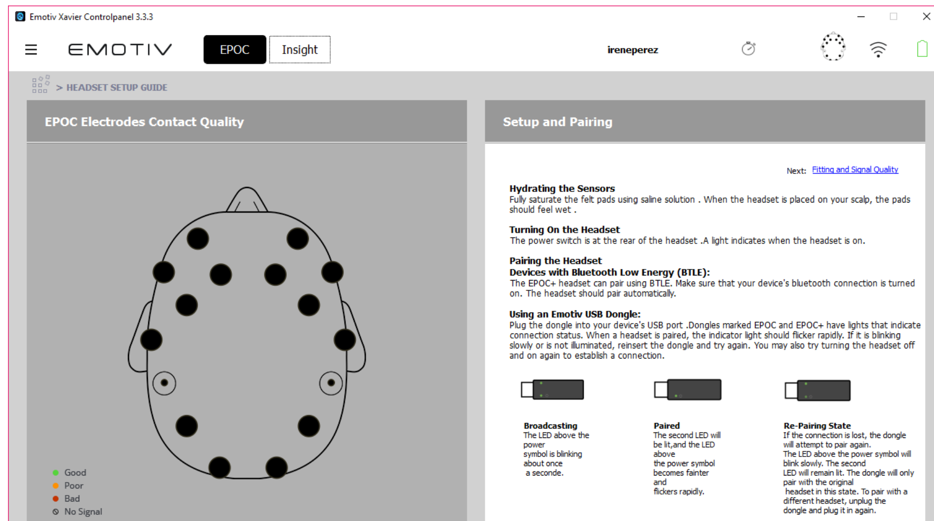


Figura 3.8 Vista principal de la aplicación ControlPanel

Al cambiar a la vista del Emotiv Insight, se encuentra la localización de los sensores correspondientes a este dispositivo (Figura 3.9).

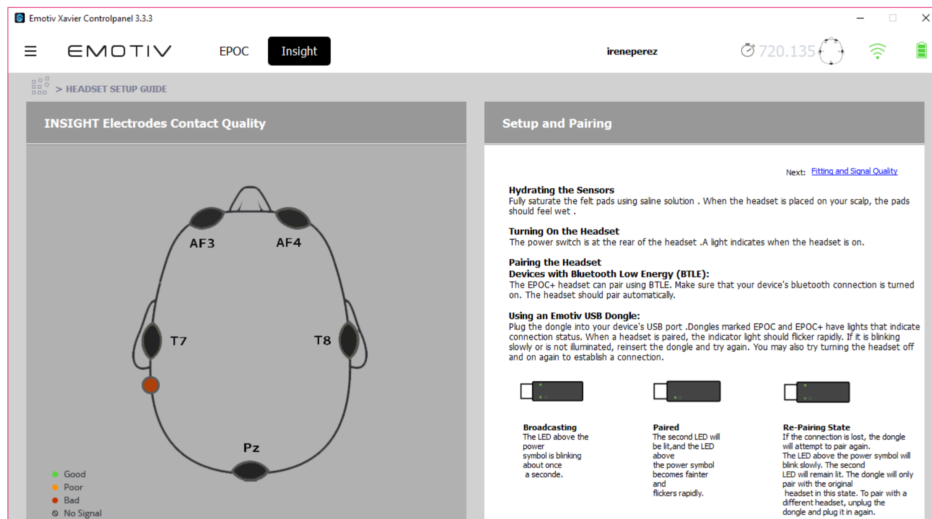


Figura 3.9 Vista Insight de la aplicación ControlPanel

Una vez se empareje con el USB, se mostrará en verde la señal de conexión, junto al indicador de batería (Figura 3.10). Si aparecen de color verde, el dispositivo está listo para usarse.

Figura 3.10 Indicadores de conexión y carga

A continuación, se debe colocar el Emotiv Insight en la cabeza. Conforme se esté haciendo, en el esquema de la cabeza que aparece en la pantalla se irán encendiendo de color verde aquellos electrodos que hagan buen contacto.

No se trata de algo inmediato, sino que hay que buscar la posición correcta moviéndolo y ajustándolo hasta conseguirlo. Una vez colocado correctamente, la pantalla principal se mostrará como en la Figura 3.11.

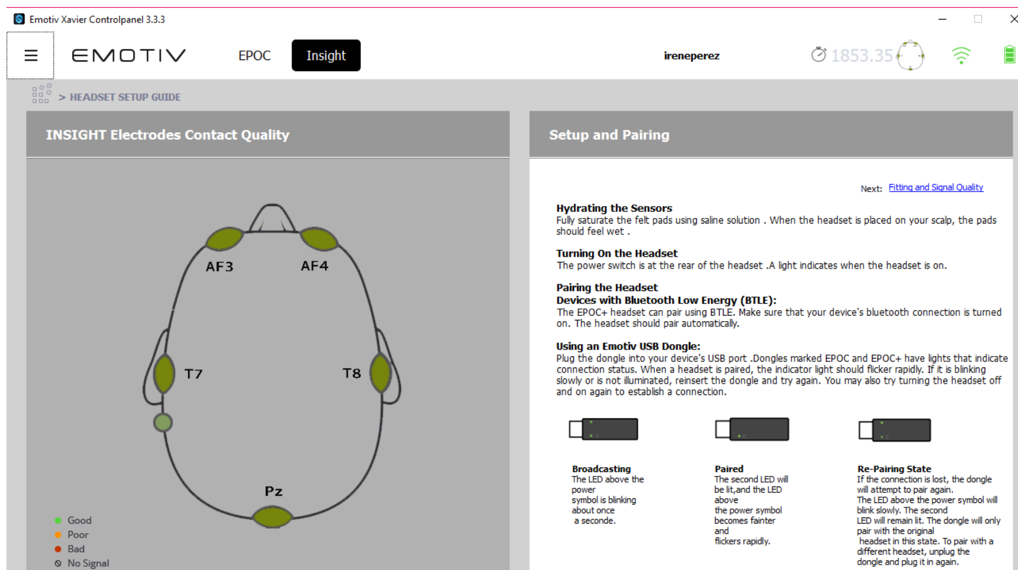


Figura 3.11 Comprobación de la buena conexión de los electrodos

3.3.3 Colocación del casco

La correcta colocación del casco en la cabeza no es un proceso trivial. Para ello, a continuación, se detallan una serie de consejos para conseguir que los electrodos hagan buen contacto fácilmente.

- Es importante colocar en primer lugar los dos electrodos de referencia (Figura 3.12), asegurando un buen contacto de éstos con la piel de detrás de la oreja. Nótese que el soporte de éstos es de goma y puede adaptarse fácilmente. En caso de no funcionar, se recomienda apretar esa zona contra la cabeza mientras se intenta colocar el resto de electrodos.

Figura 3.12 Electrodos de referencia

- Se recomienda continuar colocando los dos sensores de la frente (AF3 y AF4, Figura 3.13). Es necesario asegurarse de que tocan la piel totalmente aproximadamente unos 2 o 3 dedos por encima de las cejas.

Figura 3.13 Electrodos AF3 y AF4

Llegados a este punto, se considera que el indicador del electrodo de referencia debería ya estar de color verde, así como alguno del resto de electrodos.

- Posteriormente, se puede continuar con el sensor situado en el cuerpo del casco (T7, Figura 3.14). Necesita atravesar el pelo y tener contacto con el cuero cabelludo. Puede ayudar retirarse el pelo de la zona antes de ponerse el casco o incluso tener el cabello algo húmedo.

Figura 3.14 Electrodo T7

- A continuación, se procede a colocar el sensor que se encuentra sobre la oreja derecha (T8, Figura 3.15). Se trata de intentar que entre en contacto con la cabeza, evitando el pelo en la medida de lo posible.

Figura 3.15 Electrodo T8

- Por último, el sensor que se encuentra en la parte superior de la cabeza (Pz, Figura 3.16). Es necesario que encuentre un hueco en el que toque el cuero cabelludo. Puede ayudar separar el pelo con un peine.

Figura 3.16 Electrodo Pz

Todos estos sensores son secos, es decir, no precisan de gel, suero u otro líquido para funcionar. Sin embargo, en caso extremo de que no se consiga que hagan contacto, se puede usar suero salino o líquido de lentillas. Basta con poner unas gotas en el pelo justo donde se va a situar el sensor. Con eso debe ser suficiente, puesto que la propia transpiración y las propiedades de los sensores harán que siga haciendo un buen contacto [40].

3.3.4 Detección

El *software* de Emotiv, así como el SDK, incluye una serie de funciones para detectar determinadas expresiones y comandos mentales. A continuación, en la Tabla 3.2, se muestra una lista de las mismas [41].

Expresiones faciales	Parpadeo
	Guiño ojo izquierdo
	Guiño ojo derecho
	Fruncir ceño
	Levantar cejas
	Sonrisa
	Enseñar dientes
Comandos mentales	Neutral
	Empujar
	Tirar
	Levantar
	Bajar
	Mover a izquierda
	Mover a derecha
	Rotación horaria
	Rotación antihoraria
	Rotación hacia delante
	Rotación hacia atrás

	Rotación a la izquierda
	Rotación a la derecha
	Desaparecer
	Cualquiera definida por el usuario

Tabla 3.2 Expresiones faciales y comandos mentales

Por otro lado, el Emotiv Insight ofrece la posibilidad de tener información sobre los estados emocionales del usuario. Dichos estados están predeterminados por la aplicación y son los que se muestran en la Tabla 3.3.

Estados emocionales	Excitación instantánea
	Excitación a largo plazo
	Estrés
	Compromiso
	Relajación
	Interés / Afinidad
	Concentración

Tabla 3.3 Estados emocionales

Todas las funciones enumeradas en la Tabla 3.2 se pueden entrenar desde la aplicación Emotiv Xavier Control Panel para posteriormente usarlas en la aplicación.

3.4 Descripción del *software*

3.4.1 Emotiv Control Panel

Previamente, en este capítulo, se anticipaba uno de los posibles usos que tiene el Emotiv Control Panel al emplearlo para comprobar la conexión y el emparejamiento. Además, también se mencionaba su uso para realizar entrenamientos con el objetivo de mejorar la capacidad de detección de un usuario.

Sin embargo, esta es simplemente la primera de las funciones de este panel de control. A continuación, se realiza una descripción detallada de esta aplicación, que es de una gran importancia en el proyecto.

La Figura 3.17 muestra la pantalla principal del panel de control, que ya ha sido descrita previamente.

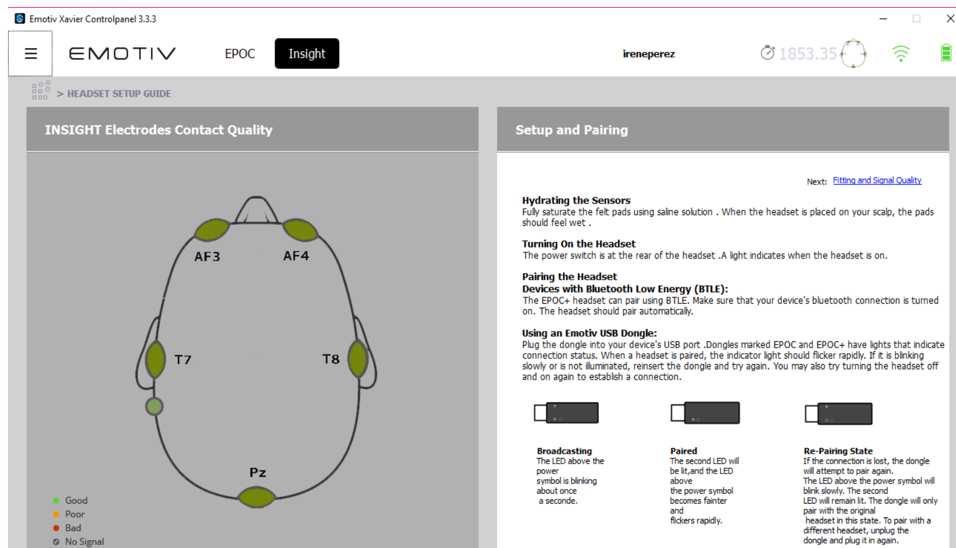


Figura 3.17 Pantalla principal del Emotiv ControlPanel

Al clicar sobre el icono que se encuentra en la esquina superior izquierda, se despliega el menú, que se muestra en la Figura 3.18.

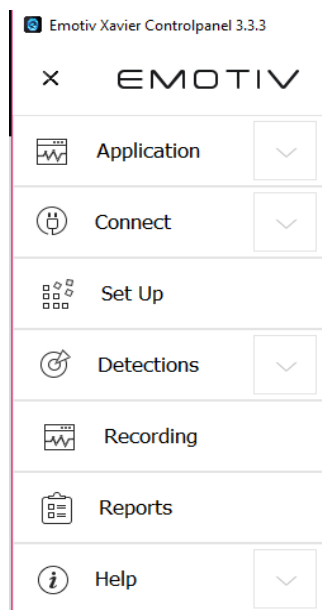


Figura 3.18 Menú del Emotiv ControlPanel

Como se puede observar, hay siete opciones a elegir: “Application”, “Connect”, “Set Up”, “Detections”, “Recording”, “Reports” y “Help”.

3.4.2 Application

“Application” (Figura 3.19) permite abrir otras aplicaciones (el Emotiv Composer y el Emotiv Emokey) y guardar una captura de pantalla del estado actual de la aplicación.

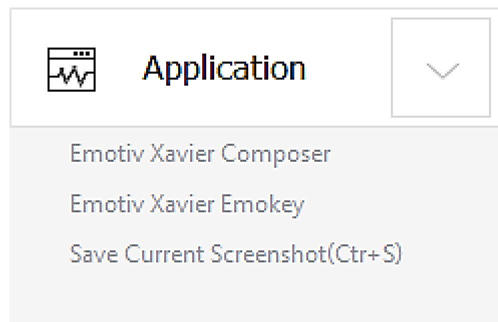


Figura 3.19 Menú del Emotiv ControlPanel: "Application"

Cabe mencionar que de estas aplicaciones se ha utilizado únicamente para el proyecto Emotiv Xavier Composer. Al tratarse de una aplicación independiente a este panel de control, se explicará posteriormente en el apartado 3.5.

3.4.3 Connect

“Connect” (Figura 3.20) permite realizar la conexión del Emotiv ControlPanel con el casco mediante “Connect Driver”, o bien con el simulador del casco mediante “Connect Composer”.

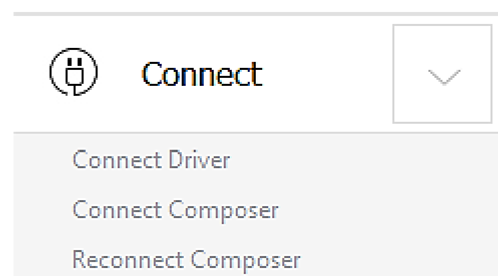


Figura 3.20 Menú del Emotiv ControlPanel: "Connect"

Por defecto, al abrir el panel de control se intentará realizar la conexión con el *driver*, por lo que no será necesario pulsar "Connect Driver".

Respecto a "Reconnect Composer", se usará cuando se pase del casco al Emotiv Composer, puesto que no se realiza de forma automática.

3.4.4 Set Up

"Set Up" se utiliza para volver a la pantalla principal mostrada en la Figura 3.17, en la que se encuentra el esquema de contacto de los sensores y la información necesaria para emparejar el dispositivo.

3.4.5 Detections

"Detections" incluye las actividades relacionadas con la adquisición de información de los sensores del casco, por lo que es especialmente importante. Tal y como se muestra en la Figura 3.21, se puede acceder a los comandos mentales, las expresiones faciales, los estados de ánimo y los sensores inerciales.

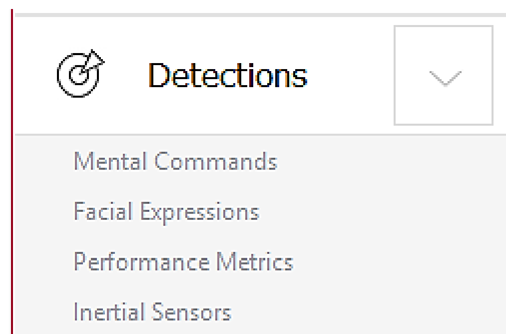


Figura 3.21 Menú del Emotiv ControlPanel: "Detections"

3.4.5.1 Mental Commands

La detección de comandos mentales evalúa la actividad cerebral del usuario en tiempo real para distinguir qué es lo que está pensando. La capacidad de detección se centra en 13 acciones diferentes:

- 6 movimientos direccionales: empujar, tirar, mover a izquierda, mover a derecha, levantar y bajar.
- 6 rotaciones: horaria, antihoraria, hacia delante, hacia atrás, hacia la izquierda y hacia la derecha.
- 1 imaginario: desaparecer.

El programa permite al usuario elegir hasta cuatro acciones que pueden ser reconocidas en un momento particular. Cuando se detecta una en un instante, se envía el identificador de dicha acción, así como un parámetro denominado potencia, que hace referencia a la correspondencia entre el estado mental actual y el entrenado.

Sin embargo, no es una tarea trivial ser capaz de controlar cuatro acciones de forma simultánea. Es recomendable comenzar con una y entrenarla hasta tener control preciso de la misma. Posteriormente, se podrán añadir otras siguiendo el mismo procedimiento.

Como forma de mostrar una realimentación al usuario, el programa dispone de un cubo 3D virtual que se mueve según el comando reconocido y que se muestra en la Figura 3.22. Dicho cubo también puede visualizarse durante el proceso de entrenamiento de una acción determinada. Además, la potencia de dicha acción se muestra en una barra marcada como “Power”.

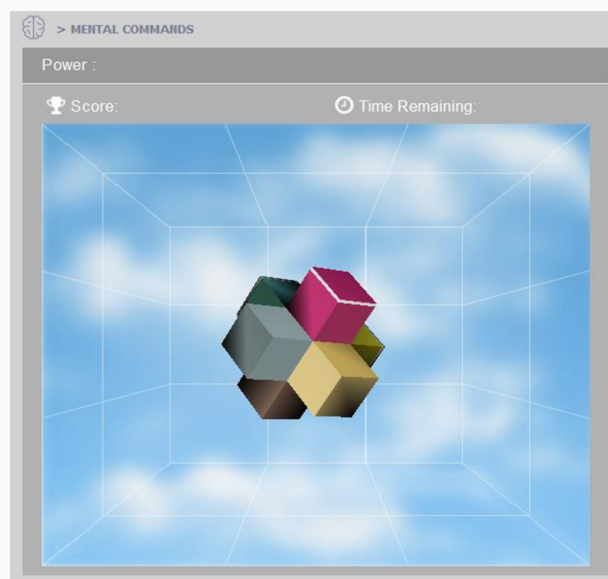


Figura 3.22 Cubo 3D para realimentación

En el lado derecho de la pantalla, junto a la pantalla de la Figura 3.22, se encuentran 4 pestañas: "Action", "Training", "Advanced" y "Settings". Esto se muestra en la Figura 3.23.

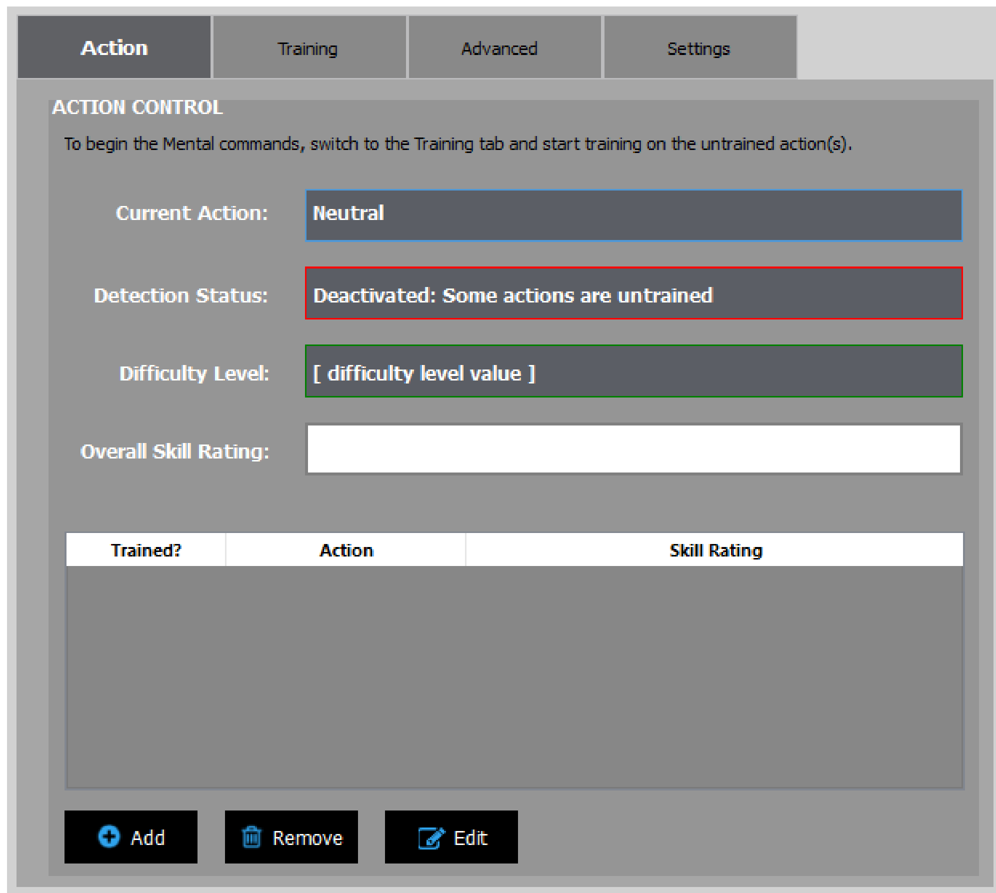


Figura 3.23 Pantalla de acción de Comando Mental

Por defecto se muestra la ventana de "Action". En ella aparece información sobre el estado actual de la detección y, además, permite al usuario definir el conjunto de acciones con las que quiere trabajar. Para poder activar la detección, es necesario haber entrenado todas las acciones elegidas además de la neutral.

El entrenamiento de cada acción se realiza de forma independiente, y cada uno de ellos tiene asociado un "Skill Rating", que también se muestra en la Figura 3.23. Este es calculado durante el proceso de entrenamiento y proporciona una medida de cómo de consistente es la capacidad del usuario para realizar adecuadamente esa acción.

Otro parámetro es el "Overall Skill Rating", que es simplemente la media de todos los "Skill Rating". Por tanto, se trata de una medida general de la habilidad del usuario para realizar el conjunto de acciones que está entrenando.

Cuando una acción se ha entrenado, aparece un indicador verde, mientras que si es rojo significa que no hay datos de entrenamiento. Es importante recordar que se debe entrenar la acción neutral, así como todas las que se desee utilizar, para poder activar la detección de comandos mentales.

El proceso de entrenamiento permite al dipositivo Emotiv analizar las ondas cerebrales del usuario y desarrollar una firma personaliza que corresponda a cada acción en concreto, así como al estado neutral. Conforme se hace la firma más precisa, la detección se vuelve más certera y fácil de ejecutar.

La pestaña de entrenamiento, que se muestra en la Figura 3.24, permite controlar el proceso de entrenamiento de los comandos mentales.

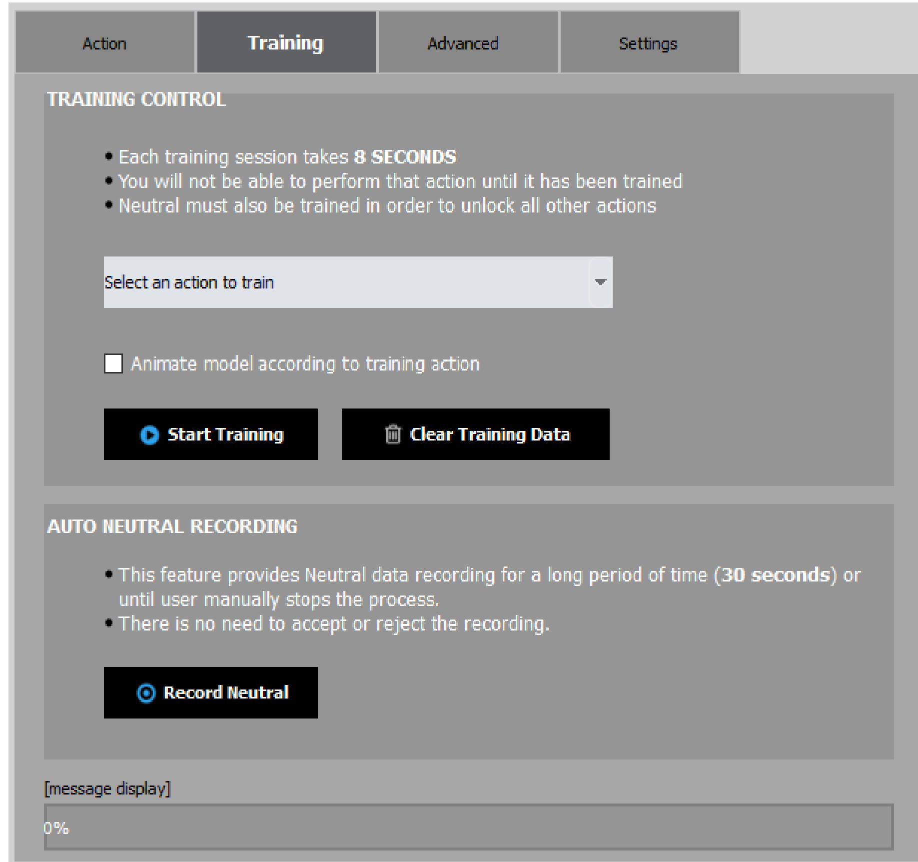


Figura 3.24 Pantalla de entrenamiento de Comando Mental

Un entrenamiento consta de tres partes:

1. Elegir una acción de la lista. Las que ya hayan sido entrenadas previamente aparecerán con un indicador verde y las que no, con uno rojo. Cabe destacar que solo las acciones que se hayan seleccionado en la pestaña “Action” aparecerán como opciones.

2. Cuando ya se tiene seleccionada la acción que se pretende entrenar, se pulsa el botón de “Start Training”. Es muy importante que, durante el tiempo del entrenamiento, se mantenga la atención en la acción. Se puede hacer uso de gestos físicos, como empujar un objeto imaginario, para reforzar el pensamiento. Sin embargo, se deben evitar los movimientos de cabeza o expresiones faciales, que pueden interferir con la señal EEG grabada

   Inicialmente, el cubo no se moverá ya que el sistema no tiene la información necesaria para construir una firma personalizada para la acción. Será una vez grabado el pensamiento neutral y habiendo entrenado la acción al menos una vez cuando se empiece a obtener la realimentación en tiempo real.

   Sin embargo, existe otro parámetro configurable por el usuario que ofrece la posibilidad de que el cubo se mueva haciendo la acción que se entrena con el objetivo de facilitar la concentración del usuario. Para ello bastará con seleccionar “Animate model according to training action”.

3. Finalmente, se debe aceptar o rechazar el entrenamiento. Esta decisión dependerá de la sensación de concentración que se haya tenido durante la sesión. También se puede abortar el entrenamiento durante el mismo

en caso de haber sido interrumpido o haber experimentado problemas con el casco.

Por otro lado, el botón “Clear Training Data” permite eliminar los datos de entrenamiento de una determinada acción. Esto es útil cuando la detección no funciona de forma adecuada para una tarea y se prefiere entrenarla de nuevo, eliminando todo entrenamiento previo.

Cabe destacar que una sesión puede ser descartada automáticamente si la señal de conexión es débil o el contacto de los sensores no es el adecuado. En este caso, se mostrará una notificación al usuario.

Tal y como se observa, una de las acciones disponibles en “Training Control” es el estado neutral. Este hace referencia al estado mental pasivo del usuario, es decir, aquel que no está asociado con ninguna de las acciones detectables. Una buena forma de entrenarlo es relajándose, o incluso leyendo. Sin embargo, debido a la importancia que presenta este estado, existe otra forma de entrenarlo. Esta surge del hecho de que es más fácil mantener la concentración en el estado neutral que otros, por lo que se permite realizar el entrenamiento durante 30 segundos. Para ello se hará uso de “Record Neutral”, que se muestra en la Figura 3.25.

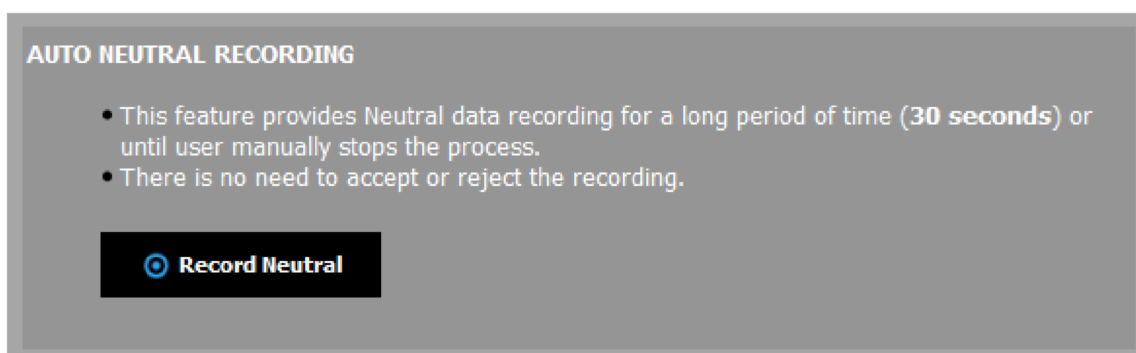


Figura 3.25 Grabación del estado neutral

La pestaña de configuración avanzada, que se muestra en la Figura 3.26, permite ajustar la sensibilidad de los comandos mentales, así como habilitar aspectos relacionados con la información almacenada en la memoria de la firma.

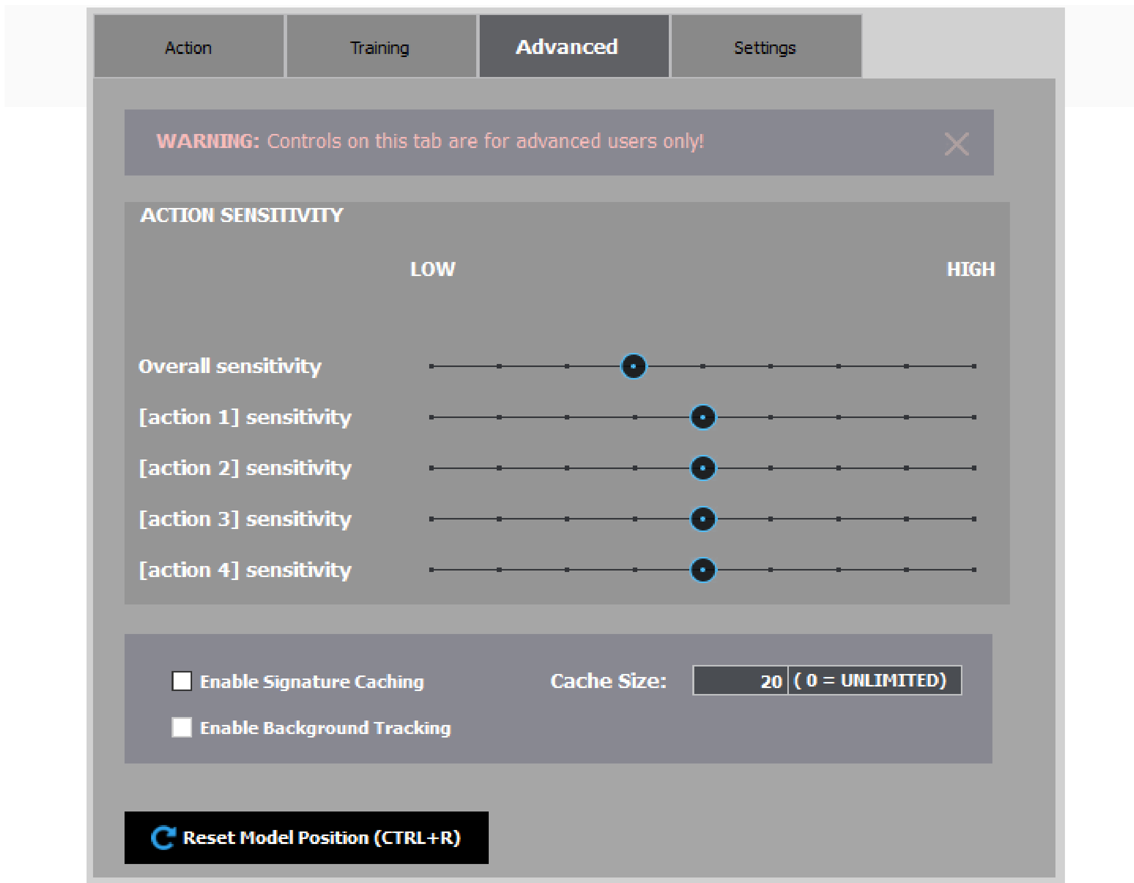


Figura 3.26 Pantalla de configuración avanzada de Comando Mental

Por defecto, la detección de comandos mentales está configurada de forma que produzca el mejor resultado posible para la mayoría de los usuarios. Por tanto, es muy recomendable que solo se cambien estos parámetros si se tiene un conocimiento profundo de Emotiv EmoEngine.

La última pestaña es la de ajustes y configuración de modelos, que se muestra en la Figura 3.27. En primer lugar, en la ventana “Model Control” se puede elegir si se desea que sea bidimensional o tridimensional, personalizar el fondo, el marco… o aplicar el escenario por defecto.

Sin embargo, hay otras dos ventanas que también incluyen aspectos útiles: “3D Settings” e “Import Page”. La primera permite elegir entre un modelo de coche y uno de cubo, así como otros que se añadan; y la segunda es la que da la posibilidad de incorporar nuevos modelos.

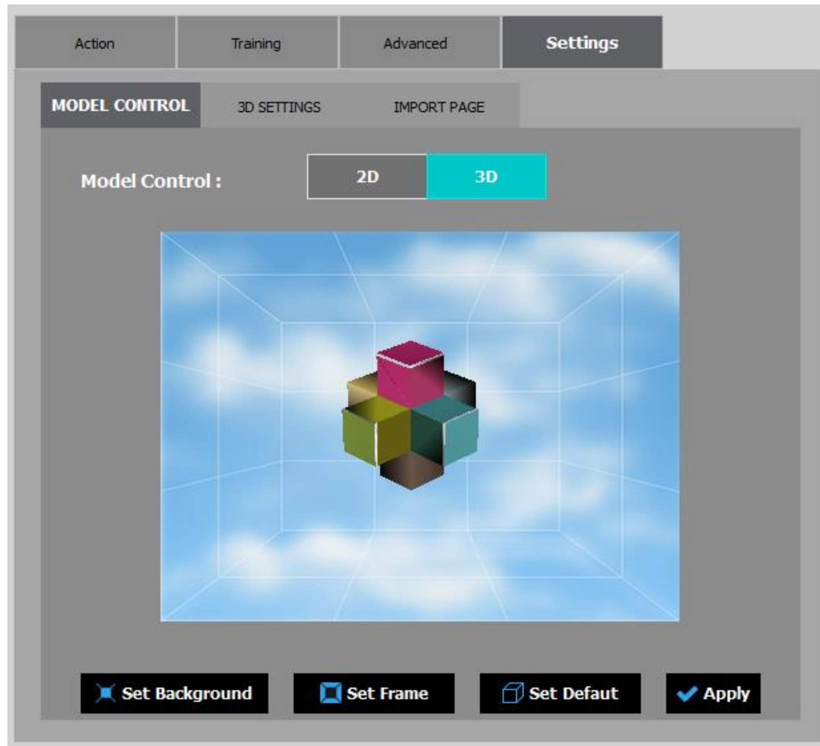


Figura 3.27 Pantalla de ajustes de Comando Mental

Tal y como se ha mencionado, se puede importar un modelo tridimensional en formato 3ds desde la ventana mostrada en la Figura 3.28. Para ello basta con añadirlo desde "Import page", moverlo por los ejes, rotarlo y escalarlo hasta situarlo donde se desee. Por último, es necesario seleccionar "Command Object" para empezar a trabajar con él. De forma añadida, si se desea volver a usar el modelo importado en sucesivos entrenamientos, se debe hacer uso de "Export Settings". Al hacerlo se guarda un archivo EMS que contiene la información de posición, textura y escala del modelo, de forma que pueda cargarse en otra ocasión.

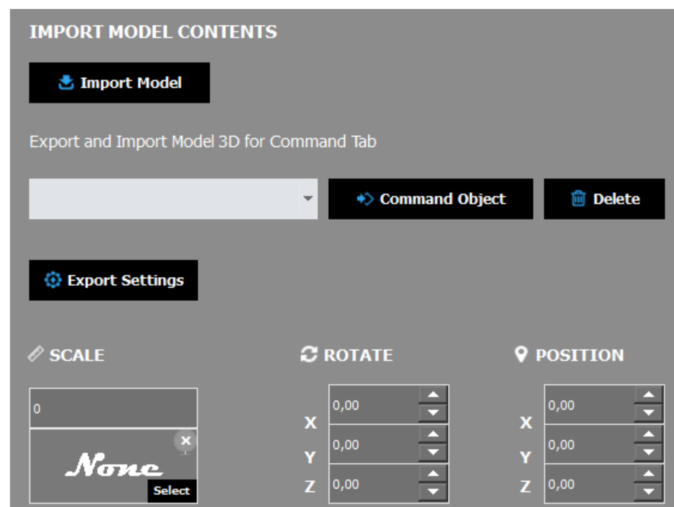


Figura 3.28 Importar modelo para entrenamiento

3.4.5.2 Facial Expresions

La detección de expresiones faciales evalúa la actividad cerebral del usuario en tiempo real para distinguir qué gesto está haciendo. La capacidad de detección se centra en 7 acciones diferentes:

- 3 asociadas a los ojos: guiño izquierdo, guiño derecho y parpadeo.
- 2 asociados a las cejas: levantar cejas (sorpresa) y bajar cejas (fruncir ceño).
- 2 asociadas a la boca: sonrisa y enseñar dientes.

El programa reconoce la expresión facial que se haga en un momento particular. Cuando se detecta una en un instante, se envía el identificador de dicha acción y un parámetro denominado potencia, que hace referencia a la correspondencia entre la expresión actual y la entrenada.

Como forma de mostrar una realimentación al usuario, el programa dispone de un avatar que representa la expresión facial y que se muestra en la Figura 3.29. Cabe destacar que se muestra como si de vista de cámara se tratara, no en espejo.

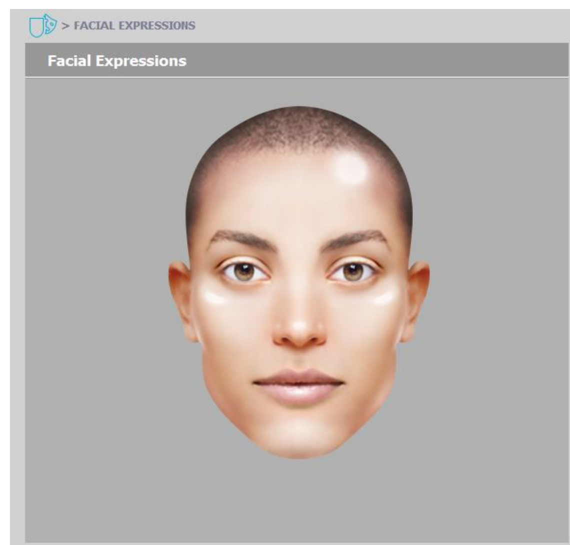


Figura 3.29 Avatar para realimentación

En el lado derecho de la pantalla, junto a la pantalla de la Figura 3.22, se encuentran 2 pestañas: "Sensitvity" y "Training".

En la Figura 3.30 se muestra la ventana de “Sensitivity”. En ella aparecen unos deslizadores que permiten ajustar la sensibilidad de cada una de las expresiones: si la detección es demasiado sensible para una determinada expresión, se debe mover el deslizador hacia valores negativos, y viceversa. A la derecha de cada deslizador aparece un indicador vertical verde que hace referencia a la potencia de detección, que se ha explicado previamente.

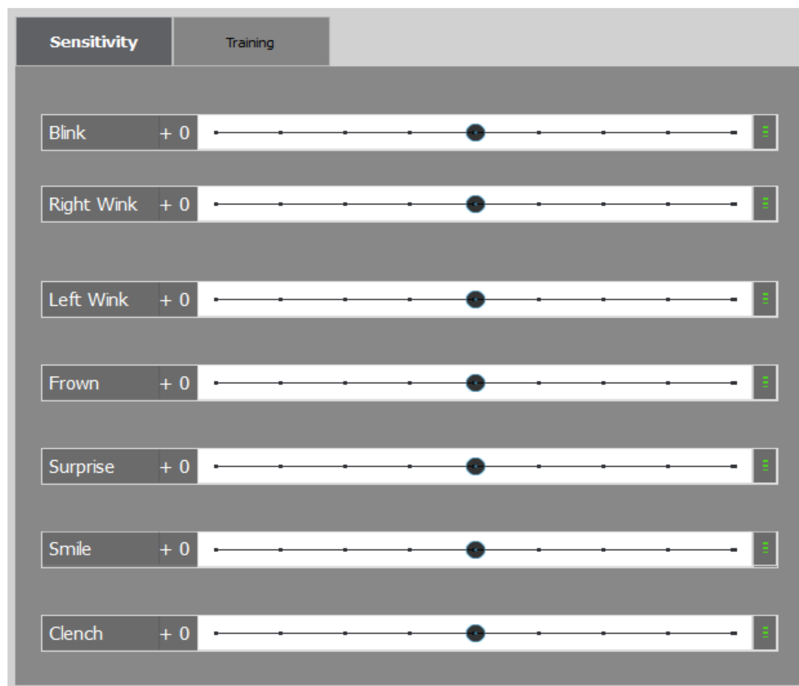


Figura 3.30 Pantalla de sensibilidad de Expresiones Faciales

La detección de expresiones faciales soporta dos tipos de firmas que permiten clasificar la acción mental y asociarla a una expresión. La firma universal está diseñada para ajustarse a gran parte de la población, pero no tiene por qué funcionar bien con cualquier usuario. Para evitar problemas, es recomendable utilizar una firma entrenada.

El entrenamiento, al igual que en el caso de la detección de comandos mentales, consiste en realizar la acción o la expresión facial con el objetivo de que se obtenga la señal EEG generada. Se realiza desde la ventana “Training”, mostrada en la Figura 3.31 y, cuanto mayor sea el entrenamiento, más preciso será el sistema. Un dato a tener en cuenta es que, si se selecciona la firma entrenada, solo se

detectarán las expresiones que se hayan entrenado. Como mínimo se debe disponer de datos de la expresión neutral y al menos una de las expresiones disponibles para entrenar (las relacionadas con las cejas y boca).

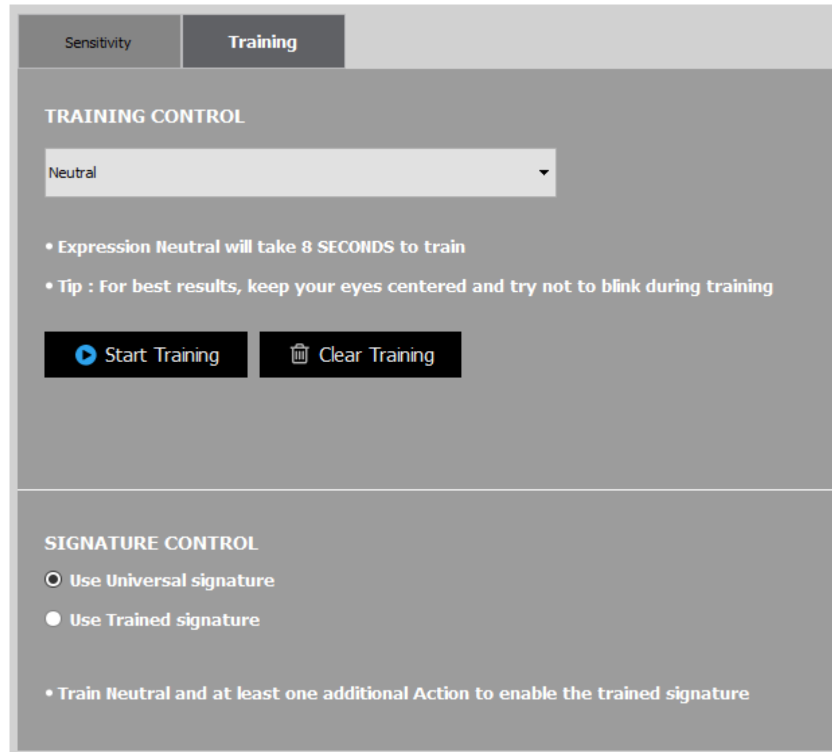


Figura 3.31 Pantalla de entrenamiento de Expresiones Faciales

3.4.5.3 Performance Metrics

En esta ventana, que se muestra en la Figura 3.32, aparecen dos gráficas que representan, en tiempo real, los cambios experimentados en ciertas emociones del usuario. Emotiv contempla 7 variables:

- Interés
- Compromiso
- Estrés
- Relajación
- Excitación instantánea
- Excitación a largo plazo
- Concentración

En este caso, se buscan señales cerebrales características que son universales y, por tanto, no requieren de entrenamiento o firma específica de un usuario. Sin embargo, se recogen los datos individuales de cada usuario y se guardan en el perfil de dicho usuario mientras se realiza el análisis. Con estos datos se consigue mejorar la precisión de la detección con el tiempo.

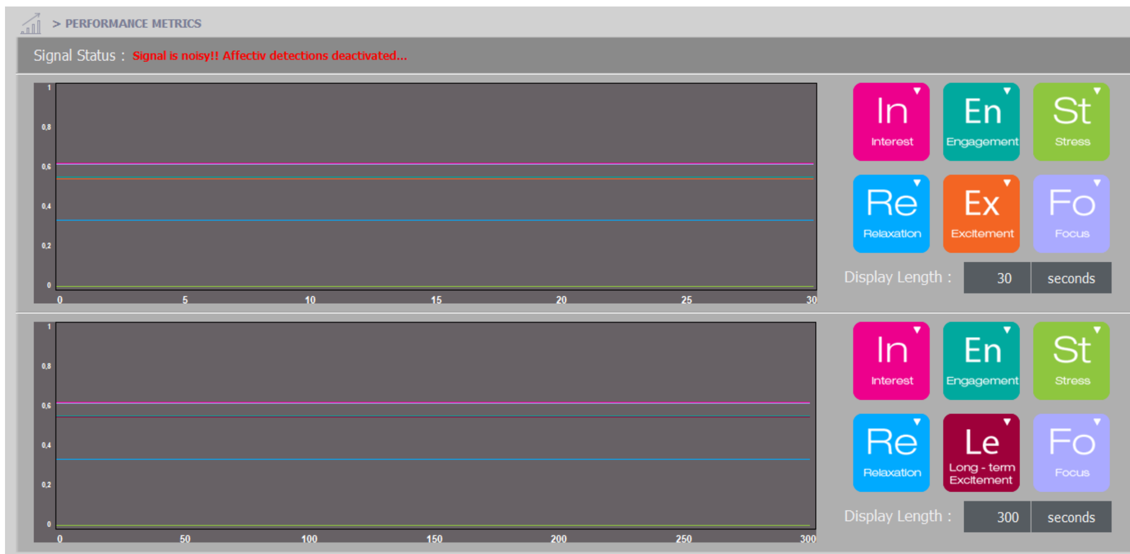


Figura 3.32 Gráficas de Estados de Ánimo

Estas gráficas pueden editarse, quitando cualquiera de los estados de ánimo, cambiando los colores e incluso la escala de tiempos.

3.4.6 Recording

"Recording", también llamado en otras versiones "Calibration", es una de las nuevas incorporaciones de Emotiv para Insight. Permite elegir un tipo de actividad, tales como conducir, dibujar, meditar, memorizar... y realizar un análisis de los estados de ánimo basándose en dos calibraciones: una con los ojos abiertos y otra con ellos cerrados.

3.4.7 Reports

"Reports" permite obtener informes con datos sobre las sesiones realizadas, de forma que se puedan emplear para hacer comparativas entre usuarios o entre

distintos entrenamientos del mismo usuario. También se trata de una función exclusa del Emotiv Insight.

3.4.8 Help

Desde "Help" (Figura 3.33) se puede acceder, en primer lugar, al repositorio de Emoitv en GitHub, donde se encuentra disponible el SDK para poder desarrollar aplicaciones propias.

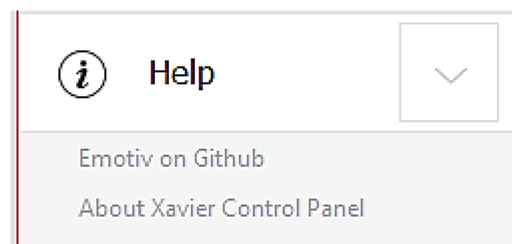


Figura 3.33 Menú del Emotiv ControlPanel: "Help"

Además, se puede obtener información acerca del panel de control instalado, la versión, dirección web y otros aspectos relativos al proyecto en el que se basaron para mostrar sus gráficas e imágenes. Esto se muestra en la Figura 3.34.

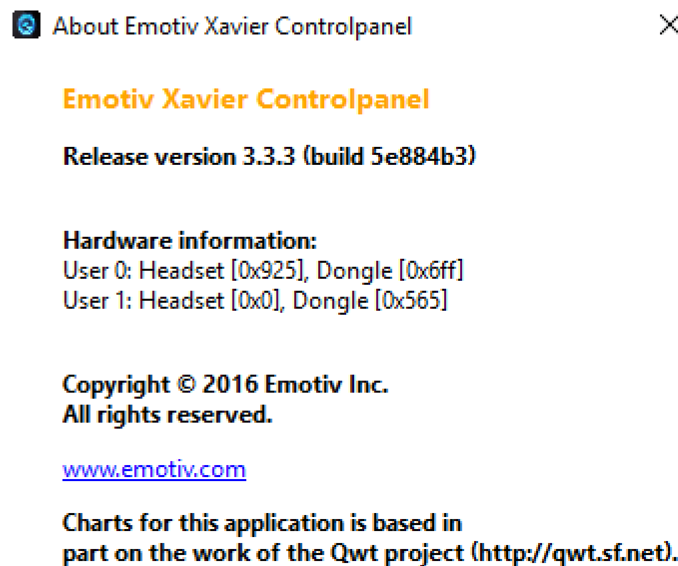


Figura 3.34 Información sobre Emotiv ControlPanel

3.5 Emotiv Composer

Emotiv Composer es un simulador del comportamiento de EmoEngine. Fue creado con el objetivo de servir como herramienta para el desarrollo y pruebas de cualquier aplicación. Facilita el proceso de programación de una aplicación puesto que evita usar el casco en muchas ocasiones mediante el envío de eventos de EmoEngine simulados.

Por defecto, esta aplicación se muestra como en la Figura 3.35. Se trata del modo interactivo, que permite definir y enviar valores específicos de EmoState a cualquier aplicación que haga uso del SDK de Emotiv.

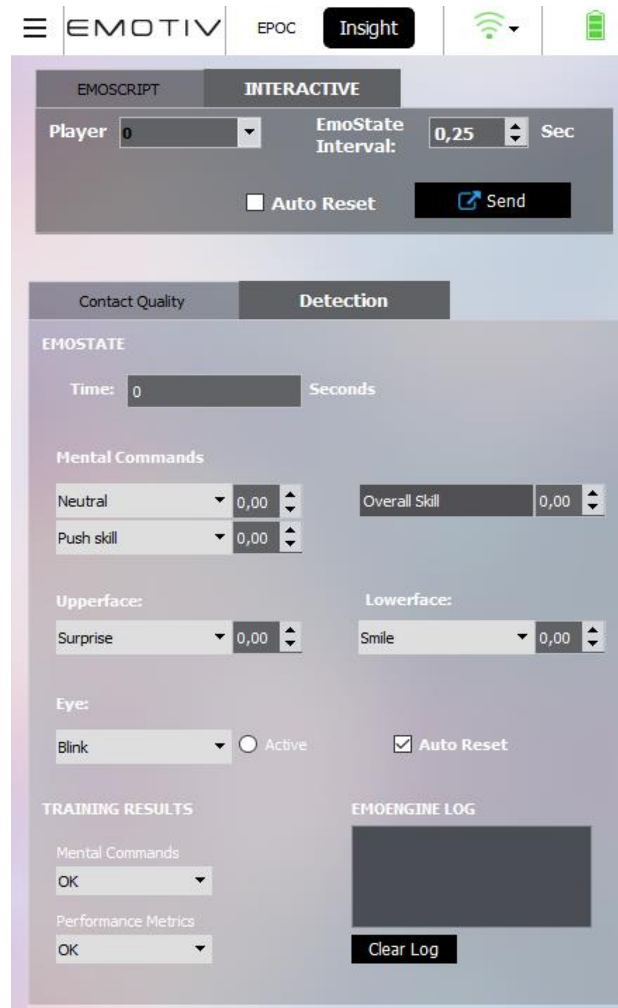


Figura 3.35 Pantalla Interactiva de Emotiv Composer

Se puede cambiar a modo "EmoScript", que se muestra en la Figura 3.36. Este permite ejecutar una secuencia predefinida de valores de estado en cualquier aplicación que use EmoEngine. Los archivos a usar se escriben en EML (Emotiv Markup Language), cuya sintaxis se encuentra en el manual de usuario.

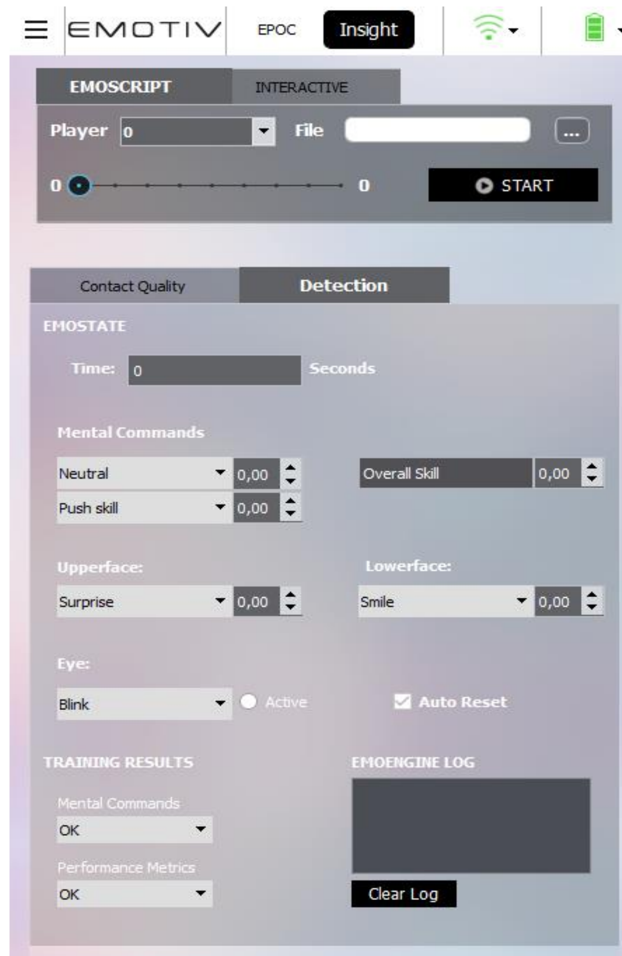


Figura 3.36 Pantalla EmoScript de Emotiv Composer

Analizando estas pantallas se observa, en primer lugar, tres partes bien diferenciadas: barra de configuración (tipo de casco, conexión y batería), superior ("EmoScrip" o "Interactive") e inferior ("Contact Quality" y "Detection").

En la barra de configuración, una vez elegido el tipo de casco a utilizar (Emotiv Insight), se puede configurar la calidad de la conexión y la batería del dispositivo.

- "Wireless": al seleccionar el icono de Wi-Fi, se permite al usuario elegir la fuerza de la conexión con el dispositivo. Esto permite simular fallos en la conexión para ver cuál es el comportamiento de la aplicación que se desarrolla ante este evento.
- "Battery": de forma análoga al icono de Wi-Fi, el icono de batería permite al usuario realizar simulaciones ante distintos estados de carga.

Bajo dicha barra, se encuentra la zona de "Interactive/EmoScript". Para el caso del modo interactivo se pueden configurar distintos parámetros:

- "Player": permite elegir el número de usuario cuyo EmoState se va a definir y enviar. Por defecto es 0. Cuando se cambia por primera vez este número, la aplicación conectada a este simulador recibe un evento de nuevo usuario.
- "EmoState Interval": es el periodo de tiempo que transcurre entre el envío de dos EmoStates.
- "Auto Reset": se debe marcar esta opción si se desea que los nuevos EmoStates se envíen automáticamente, con el período especificado en "EmoState Interval".

Por otro lado, en caso de seleccionar el modo EmoScript, se han de configurar otros parámetros:

- "Player": análogo al del modo interactivo. Por defecto es 0 y podrá elegirse entre 0 y 1.
- "File": para buscar y cargar un archivo EML del disco. Si la carga se completa correctamente, entonces la barra de tiempo que se encuentra debajo y el botón "Start" se activan. Si ocurre un error, aparecerá un mensaje con la descripción y localización del fallo en el archivo.
- "Timeline Slider": la barra de tiempos es un deslizador que permiten ver el estado y la calidad de señal en cualquier instante de tiempo definido en el archivo cargado.

Por último, bajo las ventanas de “EmoScript” e “Interactive”, se encuentran otras que son comunes y, por tanto, independientes del modo elegido. Se trata de “Contact Quality” y “Detection”.

La ventana que se muestra en la Figura 3.37 permite ajustar la calidad de cada sensor del casco (excepto los de referencia) de forma individual en caso de usar el simulador. Si, por el contrario, se está leyendo un archivo EmoScript, será un indicador de la calidad definida (por defecto, GOOD).

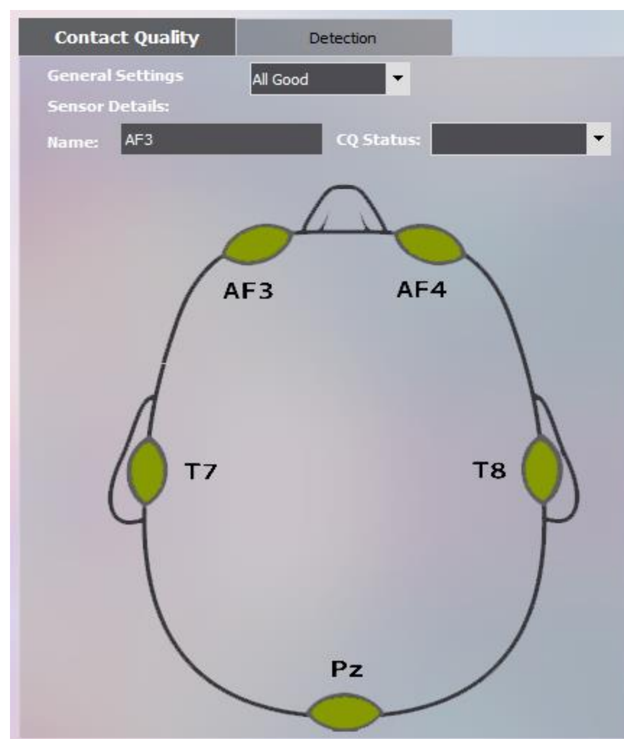


Figura 3.37 Calidad de contacto en Emotiv Composer

Al cambiar a la ventana de detección, aparecen una serie de parámetros que son los que permiten modificar y enviar los nuevos estados (en caso del modo interactivo), o visualizar los estados que se están llevando a cabo según el fichero (en caso del EmoScript).

Para comenzar a utilizarlo basta con seleccionar la acción que se quiera mandar, bien sea comando mental o expresión facial, indicar su potencia y darle a a “send”.

Por último, hay un cuadro de texto donde aparece la información de la ejecución, el “EmoEngine log”. El objetivo es ofrecer al usuario una imagen clara de cómo funciona la API de Emotiv. La información sobre los distintos mensajes que aparecen se puede encontrar en el manual de usuario que se adjunta con la presente memoria.

Capítulo 4

Kinova Mico²

4.1 Introducción

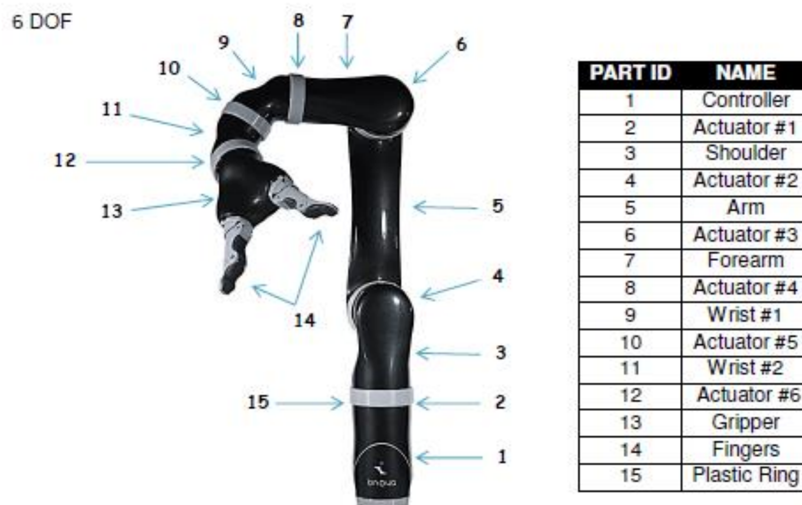
La finalidad del presente capítulo es mostrar una descripción detallada del Kinova Mico², empleado en el TFG. Por tanto, se incluye información relativa a las especificaciones físicas del mismo, así como el *software* con el que se ha usado.

4.2 Descripción del *hardware*

4.2.1 Especificaciones técnicas

El Kinova Mico² es un brazo robótico ligero compuesto de 6 segmentos interconectados. Por medio del controlador o a través del ordenador, el usuario puede mover el robot en las tres dimensiones del espacio, así como coger y soltar objetos en caso de incorporar una mano.

La Figura 4.1 muestra las distintas partes del brazo robótico que se describe, indicando también los actuadores.



PART ID	NAME
1	Controller
2	Actuator #1
3	Shoulder
4	Actuator #2
5	Arm
6	Actuator #3
7	Forearm
8	Actuator #4
9	Wrist #1
10	Actuator #5
11	Wrist #2
12	Actuator #6
13	Gripper
14	Fingers
15	Plastic Ring

Figura 4.1 Partes del Mico²

Respecto a las especificaciones técnicas generales del dispositivo, se muestran en la Tabla 4.1.

Grados de libertad	6
Ángulo de la muñeca	55º
Peso sin mano	4.6 kg
Peso mano 3 dedos	727 g
Peso total	5.4 kg
Carga media en condiciones normales	1.1 kg
Carga en funcionamiento extremo	0.6 kg
Alcance	70 cm
Consumo medio	25 W (5 W en standby)
Consumo máximo instantáneo	100 W
Temperatura de operación	-10ºC a 40ºC
Velocidad lineal máxima	20 cm/s
Protocolo de comunicación	RS-485
Cable de comunicación	Cable flexible de 20 pines
Resistencia al agua	IPX2

Tabla 4.1 Especificaciones técnicas generales del Mico²

A continuación, se muestra en la Tabla 4.2 la información relativa a la comunicación del brazo robótico, y en la figura las conexiones del Kinova Mico².

Puerto Joystick	1 Mbps Canbus
Puerto Alimentación	18 a 29 VDC, 24 VDC nominal
Puerto USB 2.0	12 Mbps
Puerto Ethernet	100 Mbps
Frecuencia del sistema de control alto nivel	100 Hz
Frecuencia del sistema de control bajo nivel	500 Hz
Frecuencia CPU	360 MHz
Control	Fuerza, coordenadas cartesianas y angulares

Tabla 4.2 Especificaciones de comunicación del Mico²

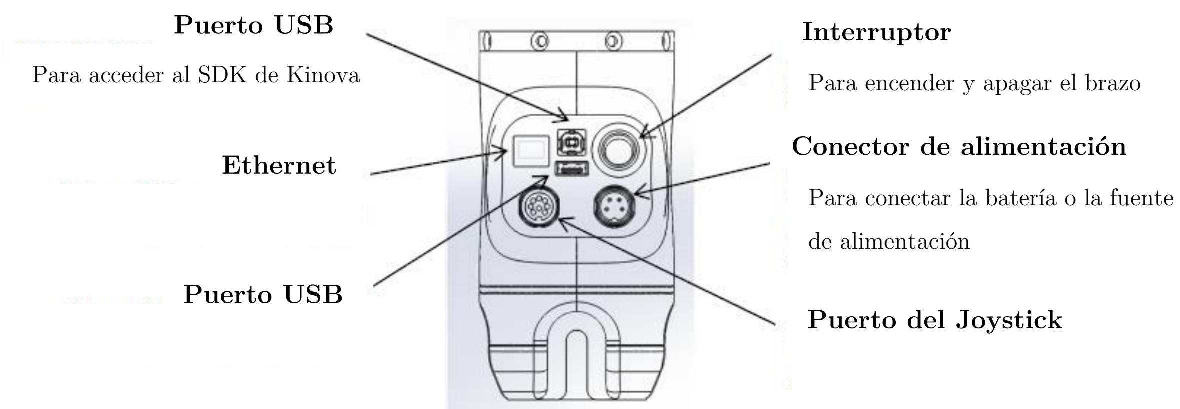


Figura 4.2 Conexiones del Mico²

Por otro lado, tal y como se ha mencionado, la mano que se ha usado es la de 3 dedos. Sus especificaciones son las que se muestran en la Tabla 4.3.

Número de dedos	3
Número de actuadores	Uno por dedo
Sensores	Corriente
	Temperatura

	Encóder rotacional
Fuerza de agarre	25 N
Tiempo de apertura y cierre	1.2 s

Tabla 4.3 Especificaciones técnicas de la mano

4.2.2 Puesta en marcha

La instalación del Mico^2 incluyen cuatro pasos, que se describen a continuación.

1. Integración mecánica: el brazo ha sido diseñado para instalarse sobre una superficie fija o una plataforma móvil. Es necesario asegurarse de que el brazo está fijo y que la base no se va a caer durante operaciones que impliquen llevar al alcance máximo del robot.

   Para el proyecto, se montó el brazo sobre la base y se sujetó a la mesa tal y como se muestra en la Figura 4.3.

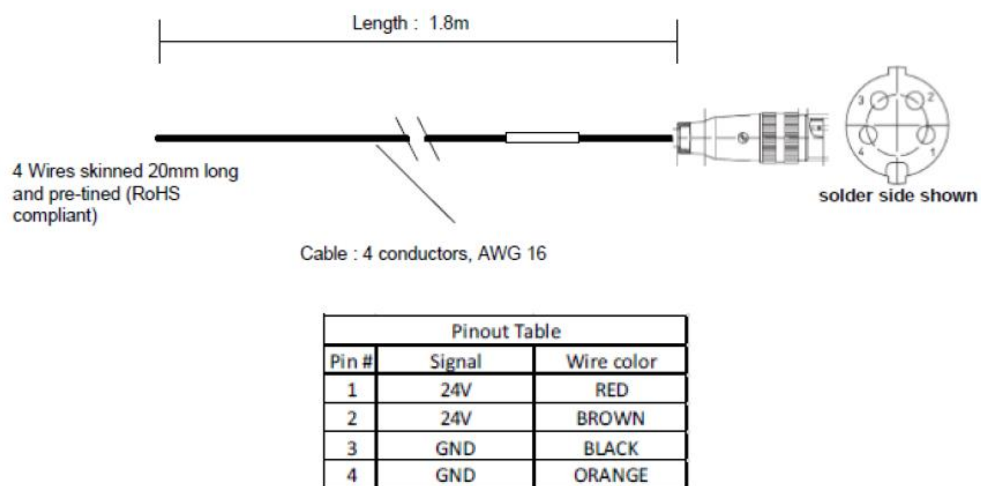
Figura 4.3 Sujeción del brazo a la mesa

2. Integración eléctrica: hay dos formas de alimentar el Mico^2: mediante conexión a la red eléctrica o con batería.

Si se va a dejar el brazo fijo en un lugar con acceso a la red, como es el caso del presente proyecto, se puede usar una salida de tensión 110/220 V estándar conectando la fuente que se muestra en la figura.

Figura 4.4 Alimentación del Mico²

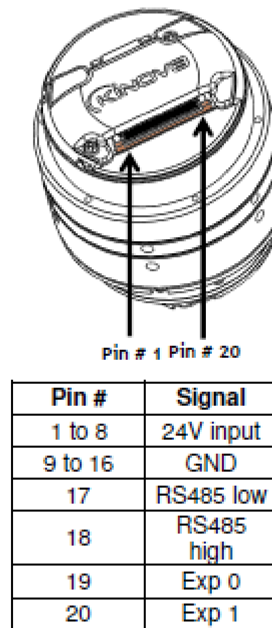
En cambio, si se necesita dotar al sistema de cierta portabilidad e independencia de la red eléctrica se puede usar una batería de 24V, usando las conexiones que se muestran en la Figura 4.5.



Pinout Table		
Pin #	Signal	Wire color
1	24V	RED
2	24V	BROWN
3	GND	BLACK
4	GND	ORANGE

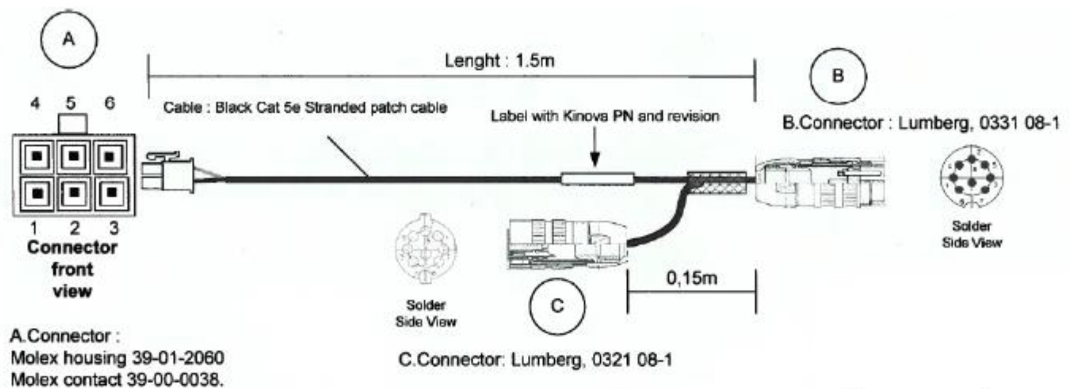
Figura 4.5 Conexión de Mico² con batería

3. Integración eléctrica de la mano: El Mico^2 tiene dos líneas de comunicación y de alimentación accesibles desde el último actuador para añadir un dispositivo adicional. Se debe efectuar la conexión de entrada (Figura 4.6) y salida (Figura 4.7).



Pin #	Signal
1 to 8	24V input
9 to 16	GND
17	RS485 low
18	RS485 high
19	Exp 0
20	Exp 1

Figura 4.6 Conexión del último actuador



Connector A		
#	Signal	Function
1	COM1	RS485_low
2	GND	GND
3	COM3	Exp 0
4	COM2	RS485_high
5	24V*	24V
6	COM4	Exp 1

*Maximum current 1,5 A

Figura 4.7 Conexión de la mano

4. Integración del control: se refiere tanto al joystick (Figura 4.8) como a la API. Para usar la API basta con conectar el USB en el puerto correspondiente del Mico². Posteriormente, es necesario instalar y abrir el SDK y seguir el procedimiento que se indica en la documentación para ponerlo en marcha. Por otro lado, para usar el joystick es suficiente con conectarlo al puerto adecuado.

Figura 4.8
Joystick del Mico²

4.2.3 Principios de operación

El control general del robot se realiza mediante el joystick que se ha mostrado en la Figura 4.8. Desde él se puede llevar al robot a dos posiciones importantes: *home* y *retracted*.

- La posición *home* (Figura 4.9) es desde la que se realizan los movimientos del robot en caso de querer moverlo a una posición requerida. Al encenderlo, el robot pasa automáticamente a esta posición y permanece a la espera de recibir un comando del mando.

Figura 4.9 Posición home

La posición *retracted* (Figura 4.10) hace referencia a la posición en la que se debe dejar el brazo cuando no se esté usando. Reduce el volumen ocupado por el brazo y permite desconectar los motores con seguridad, evitando caídas. Además, el sistema permanece en modo *standby*.

Figura 4.10 Posición retracted

4.3 Descripción del *software*

Kinova ofrece una API con programas de ejemplo en los que se puede probar su funcionamiento, pero no han sido necesarios en este proyecto. Esto se debe a que lo que se busca en este TFG es la integración en ROS de los distintos elementos.

En el caso de Emotiv, ROS es un mero sistema que le permite publicar mensajes en un *topic* para que los reciba el robot pero, para Kinova, ROS es de mucha más importancia: el control del robot se realiza por medio de un paquete desarrollado por Kinova para ROS (kinova-ros) y que incluye desde los *drivers* hasta su simulador en Gazebo.

Por tanto, esta es la razón por la que se incluye ROS como *software* de Kinova, ya que se encarga de gestionar todo el robot mediante sus nodos y *topics*.

4.3.1 ROS

ROS, tal y como se anticipó en el Capítulo 2, es el middleware robótico en torno al cual se integra el presente proyecto. Se trata de un *framework* flexible y distribuido de procesos (nodos) que permite a los ejecutables ser diseñados de manera individual. Además, estos se acoplan de manera flexible en tiempo de ejecución y se pueden agrupar en paquetes y pilas, que pueden ser fácilmente compartidos y distribuidos [42].

ROS también es compatible con un sistema federado de repositorios de código que permite la colaboración y distribución. Este diseño desde el nivel del sistema de archivos hasta el nivel de la comunidad de usuarios permite la toma de decisiones independientes sobre el desarrollo e implementación gracias a la infraestructura de herramientas de ROS.

Con el objetivo de crear *software* robótico complejo y robusto que funcione en diferentes plataformas, ROS consta de una colección de herramientas y librerías. Además, fomento el desarrollo de *software* colaborativo mediante sus foros y repositorios.

El objetivo principal de ROS es, por tanto, dar soporte a la comunidad de usuarios y permitir la reutilización de código para el desarrollo e investigación de la robótica. Aparte de este objetivo de compartir y colaborar, hay otros objetivos destacables [43]:

- ROS está diseñado para ser lo más ligero posible, de forma que el código escrito para ROS se pueda utilizar con otros *frameworks* de desarrollo robótico tales como OpenRAVE, Orocos, and Player.
- ROS dota al sistema de independencia del lenguaje puesto que ya está implementado en Python, C++ y Lisp, y existen librerías experimentales en Java y Lua.
- ROS permite realizar pruebas de forma sencilla mediante un *framework* de test llamado rostest.
- ROS es altamente escalable, lo que lo hace apropiado para sistemas de largo tiempo de ejecución y para grandes procesos de desarrollo.

4.3.1.1 Arquitectura de ROS

ROS se organiza en tres niveles conceptuales [44]:

- Nivel de sistema de archivos: cubre los recursos ROS que están almacenados en el disco, tales como paquetes, metapaquetes, repositorios, tipos de mensaje…
- Nivel gráfico de computación: es la red de procesos ROS que procesan datos juntos en configuración punto a punto. Se trata, por tanto, de nodos, mensajes, servicios, topics…
- Nivel de comunidad: son los recursos ROS que permiten que, comunidades diferentes, intercambien *software* y conocimiento. Se incluyen aquí las distribuciones, repositorios, el wiki de ROS, blogs…

Para llevar a cabo la programación es necesario dominar, principalmente, el gráfico computacional. Hay algunos conceptos que son de especial relevancia para el correcto desarrollo del proyecto, y se explican a continuación:

- Nodo: un nodo es un proceso que realiza un cómputo. Está escrito usando una librería cliente de ROS tal como roscpp (C++) o rospy (Python).
- Master: es el encargado de buscar al resto de elementos. Sin el Master, los nodos no podrían encontrarse entre si, ni intercambiar mensajes. Como parte del Master se encuentra el servidor de parámetros, que permite guardar datos mediante clave en su ubicación.
- Mensajes: es el medio de comunicación entre nodos. Se trata simplemente de una estructura de datos que contiene campos tipados.
- *Topics*: un nodo envía un mensaje al publicarlo en un *topic*. El *topic* se usa para identificar el contenido del mensaje, por tanto, un nodo que está interesado en un cierto tipo de información se suscribirá al topic que se use para ese tipo de dato.
- Servicios: se usan cuando el modelo publicador/suscriptor no es apropiado. Este es el caso de las interacciones solicitud/respuesta, que se requieren normalmente en sistemas distribuidos. Los servicios se definen como un par de mensajes: uno para la petición y otro para la respuesta.

En la Figura 4.11 se muestra un esquema básico que muestra la diferencia entre los elementos previamente mencionados.

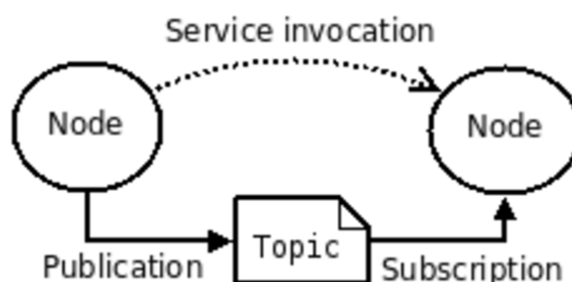


Figura 4.11 Esquema básico del envío de mensajes ROS

4.3.1.2 Gazebo

Gazebo (Figura 4.12) es el simulador que ROS provee como oficial. Se trata de un simulador que permite comprobar los algoritmos, diseñar robots, realizar comprobaciones, añadir escenarios realistas… en definitiva, probar los sistemas completos. Además, al tener físicas de sólido rígido, permite la detección de colisiones o la interacción entre distintos objetos en el entorno.

Figura 4.12 Icono de Gazebo

Una de las grandes ventajas de Gazebo es que los topics en los que publicar para el simulador son los mismos que los reales. Esto permite que, en caso de desarrollar un código, se puede probar previamente en el simulador y luego en el robot real, sin cambiar el código.

Por otro lado, lanzar Gazebo es un proceso sencillo que se hace desde terminal en Linux mediante roslaunch.

4.3.2 Herramientas software

Además de ROS y el *software* propio de Emotiv que se ha descrito en el Capítulo 3, hay otros programas que se han usado para el proyecto. Se incluyen en este apartado, aunque han sido empleados para la realización de muchos aspectos del TFG, no solo para el Kinova.

4.3.2.1 Code::Blocks

CodeBlocks (Figura 4.13) es un IDE gratuito de C, C++ y Fortran diseñado para ser extensible y completamente configurable. Se puede ampliar mediante *plugins*, que añaden nuevas funcionalidades.

Figura 4.13 Icono de Code::Blocks

Está basado en la plataforma de interfaces gráficas WxWidgets, por lo que puede usarse libremente en diversos sistemas operativos y está licenciado bajo licencia pública general de GNU.

Este entorno de programación ha sido usado para la programación del *software* de Emotiv y de las interfaces gráficas, para lo cual se añadieron sus librerías. El proceso de preparación del entorno y el uso que se ha dado a esta herramienta se explican en el Capítulo 5.

4.3.2.2 CMake

CMake (Figura 4.14) es una familia de herramientas de código abierto y plataforma cruzada diseñada para compilar, probar y empaquetar *software*. Se usa para controlar el proceso de compilación usando archivos de configuración sencillos e independientes del compilador y de la plataforma, y como resultado generar *makefiles* y espacios de trabajo que puedan ser usados en el entorno del compilador que se elija.

Figura 4.14 Icono de CMake

4.3.2.3 Catkin

Catkin es el sistema oficial de compilación de ROS, sucesor del original rosbuild. Catkin combina macros CMake y ficheros Python para dotar de cierta funcionalidad al flujo de trabajo de CMake. Sin embargo, el hecho de que esté basado en CMake hace que su funcionamiento sea similar, aunque añade soporte para buscar paquetes de forma automática y compilar varios proyectos dependientes simultáneamente.

El objetivo de Catkin es permitir una mejor distribución de paquetes, mejor compilación cruzada y mejor portabilidad que su predecesor rosbuild.

Cabe destacar que, tanto CMake como Catkin, se han usado desde ROS y en terminal, por lo que no es necesario explicar ninguna interfaz gráfica. El uso de estos programas en el proyecto se detallará en el capítulo posterior.

4.3.2.4 wxWidgets

WxWidgets (Figura 4.15) es una librería de C++ multiplataforma y libre para el desarrollo de interfaces gráficas. Se puede utilizar desde Codeblocks, de forma que, al elegir el proyecto a crear, se selecciona wxWidgets.

Figura 4.15 Icono de wxWidgets

Desde Codeblocks se puede programar usando el código propio de wxWidgets o mediante wxSmith. Esta última opción permite realizar una programación gráfica, diseñando la interfaz y programando cada elemento mediante las funciones predeterminadas.

Capítulo 5

Preparación del entorno y puesta en marcha

5.1 Introducción

En este capítulo se abordarán todos los aspectos relativos a la preparación del entorno del proyecto y puesta en marcha del sistema. Tal y como se ha explicado en los capítulos previos, la plataforma de experimentación BCI está compuesta por distintos elementos *hardware* y *software* que se encuentran convenientemente integrados. Para ello, todos los elementos se han preparado y configurado tal y como se describe a lo largo del presente capítulo.

Respecto a la preparación del *software*, teniendo en cuenta la envergadura del proyecto en cuanto a sistemas operativos utilizados, así como el uso de ROS, Gazebo, la instalación de los SDK… se ha optado por omitir la documentación de los procesos de instalación. Se explica, por tanto, la configuración de cada elemento y los aspectos importantes a tener a cuenta para el correcto funcionamiento.

Por otro lado, respecto al *hardware*, previamente se realizó la descripción de los elementos, donde se incluye la información de las conexiones y del ensamblaje. En este capítulo se añade la puesta en marcha del casco y el brazo robótico.

5.2 Preparación del *software*

5.2.1 Ubuntu

La elección de Linux para el proyecto se fundamenta en las numerosas ventajas que ofrece el poder usar ROS. Tal y como se anticipó en el Capítulo 1, ha sido necesario hacer uso de dos ordenadores diferentes para el proyecto, cada uno de ellos trabajando con una versión diferente de Linux.

La decisión de la versión de Ubuntu a instalar se ha basado en los requerimientos del *hardware*:

- El paquete para ROS de Kinova funciona con ROS Indigo en Ubuntu 14.04 [45].
- El Emotiv ControlPanel, según sus especificaciones, funciona en Ubuntu 12.04 o superior [46].

Sin embargo, tras la instalación del panel de control de Emotiv en Ubuntu 14.04, se comprobó que el programa no se ejecutaba correctamente. Al hacer llegar el problema al equipo técnico de Emotiv, estos afirmaron que la versión recomendada era Ubuntu 16.04 y que intentarían solucionarlo para posteriores actualizaciones de la aplicación.

Consecuentemente, se hizo uso de dos ordenadores, uno con Ubuntu 14.04 desde el que se controlaba el Kinova y otro con Ubuntu 16.04 para el Emotiv.

Para instalar Ubuntu en un ordenador basta con hacer una partición del disco donde instalar la imagen del sistema operativo previamente quemada en un USB. Una vez instalado se podrá arrancar el sistema y realizar las actualizaciones pertinentes. No se considera necesario incluir una descripción detallada del proceso

de instalación puesto que se aleja de los objetivos del TFG. Además, hay diversas formas de proceder, incluso se plantea la posibilidad de utilizar una máquina virtual en caso de que no se pueda hacer una partición, aunque no se ha probado su funcionamiento.

5.2.2 SDK Emotiv

Una vez instalado Ubuntu 16.04 y para poder programar la aplicación de Emotiv, es necesario instalar el SDK. Para ello, en primer lugar, hay que descargarlo desde su GitHub (https://github.com/Emotiv/community-sdk). No es necesario clonarlo en un directorio, sino que basta con descargar la carpeta y descomprimirla.

A continuación, se ha de realizar la instalación de las reglas udev de Emotiv. Udev es un gestor de dispositivos para núcleos Linux que se encarga, entre otras cosas, de mantener actualizado el directorio /dev con los dispositivos presentes del sistema. La forma en la que udev gestiona los dispositivos del sistema es a través de reglas, que se leen del directorio /etc/udev/rules.d. Deben tener, al menos, permisos de lectura para el superusuario [47].

Para instalar las reglas basta con copiar la carpeta /etc/udev del SDK en la carpeta /etc/udev del sistema. Para ello es necesario hacer uso de los permisos de superusuario:

```
$ sudo nautilus
```

Al ejecutar el comando mostrado arriba, se abrirá un explorador de archivos en el que se podrán copiar las reglas tal y como se ha mencionado previamente.

También bajo permisos de superusuario, se han de copiar todos los ficheros de /bin/linux64 en la carpeta del sistema /usr/local/lib para añadir al sistema las librerías de Emotiv.

Por último, falta copiar la carpeta /include del SDK en /usr/local/include. Para una mejor organización, se ha creado una carpeta “emotiv”, donde se incluyen

en el sistema las cabeceras necesarias para desarrollar aplicaciones de Emotiv. Tras esto, se daría por finalizado el proceso de instalación del SDK.

Aparte del SDK, es necesario instalar dos paquetes adicionales de “libdbus”.

```
$ sudo apt-get install libdbus-1-3
$ sudo apt-get install libdbus-1-dev
```

Libdbus es una librería que permite intercambiar mensajes a través de dBus, que es un sistema de comunicación entre procesos. Concretamente, esta librería permite notificar cambios *hardware* o *software*. Se utiliza para obtener mensajes de Bluetooth.

Una vez realizados los cambios indicados, ya se pueden compilar los programas desarrollados o los ejemplos del SDK.

5.2.2.1 Ejemplos del SDK

Para poder utilizar y ejecutar los ejemplos que incluye el SDK de Emotiv es necesario compilar con gcc usando los parámetros necesarios.

Por ejemplo, para compilar el ejemplo “GyroData” que se encuentra en el SDK en /examples_basic/C++ habría que escribir en la terminal lo siguiente:

```
$ cd community-sdk-master/examples_basic/C++/GyroData
$ g++ -std=c++11 main.cpp -o main -ledk -ldbus-1
-I/usr/local/include/emotiv
```

Por otro lado, para compilar los ejemplos que incluyen una interfaz gráfica es necesario añadir ciertos parámetros. Por ejemplo, para compilar “FacialExpressionDemo” se pondrá:

```
$ cd community-sdk-
master/examples_basic/C++/FacialExpressionDemo
$ g++ -std=c++11 FacialExpressionDemo.cpp -o main -ledk
-lGL -lGLU -lglut -ldbus-1 -I/usr/local/include/emotiv
```

5.2.3 Kinova-api

Para poder conectar el robot Kinova al ordenador con Ubuntu 14.04 es necesario instalar el *driver* de Kinova. Este se encuentra en el dispositivo USB de Kinova que incluye el SDK, ejemplos, drivers para distintos sistemas operativos... Concretamente, se debe ejecutar el controlador que se encuentra en Driver/Ubuntu/64bits. Al hacerlo, se abre una ventana del Centro de *Software* de Ubuntu.

Desde ella se descarga e instala el paquete Kinova-api. Puede aparecer el mensaje que se muestra en la Figura 5.1, pero se ignora mediante "Ignorar e instalar".

Figura 5.1 Mensaje de instalación de Kinova-api

Finalmente, la pantalla se mostrará como en la figura.

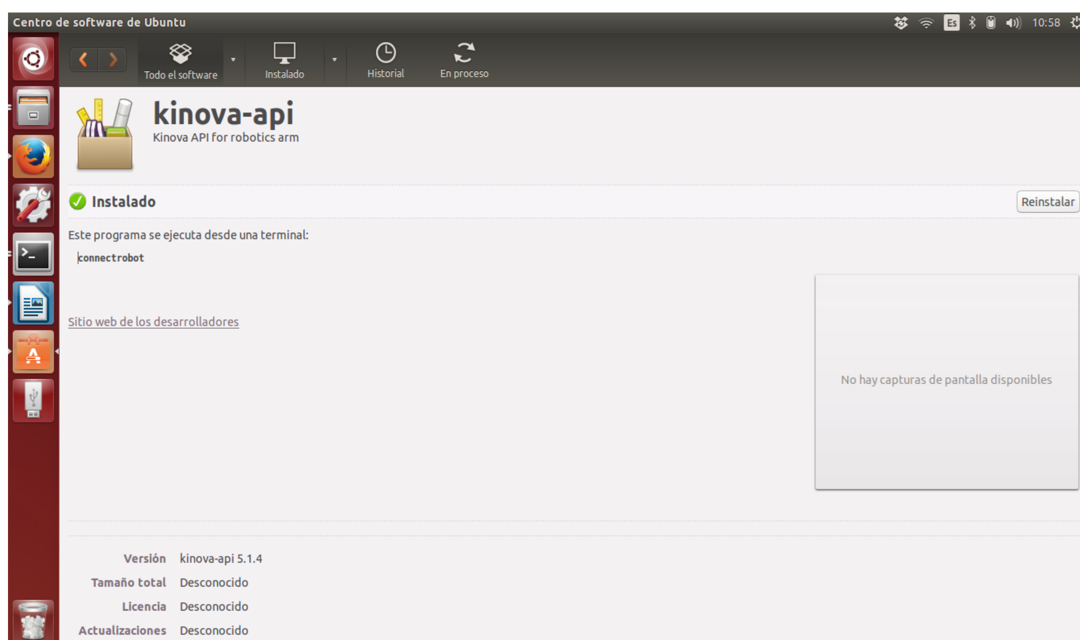


Figura 5.2 Instalación exitosa de Kinova-api

5.2.4 Code::Blocks

Para realizar la programación de Emotiv se ha hecho uso del IDE Code::Blocks (de aquí en adelante Codeblocks). Por tanto, para poder compilar el código desarrollado es necesario realizar un proceso de configuración del entorno de programación.

En primer lugar, al abrir Codeblocks es necesario crear un nuevo proyecto. Al seleccionar "Create New Project", se elige "Console Application", posteriormente el lenguaje C++ y, finalmente, se le da nombre y localización. Por su lado, el compilador se deja con la configuración por defecto.

Una vez creado, se selecciona Project/Properties… del menú superior. En el apartado "C/C++ parser options" se añade la ruta de include (Figura 5.3), lo que permitirá hacer uso del autocompletado al escribir el código.

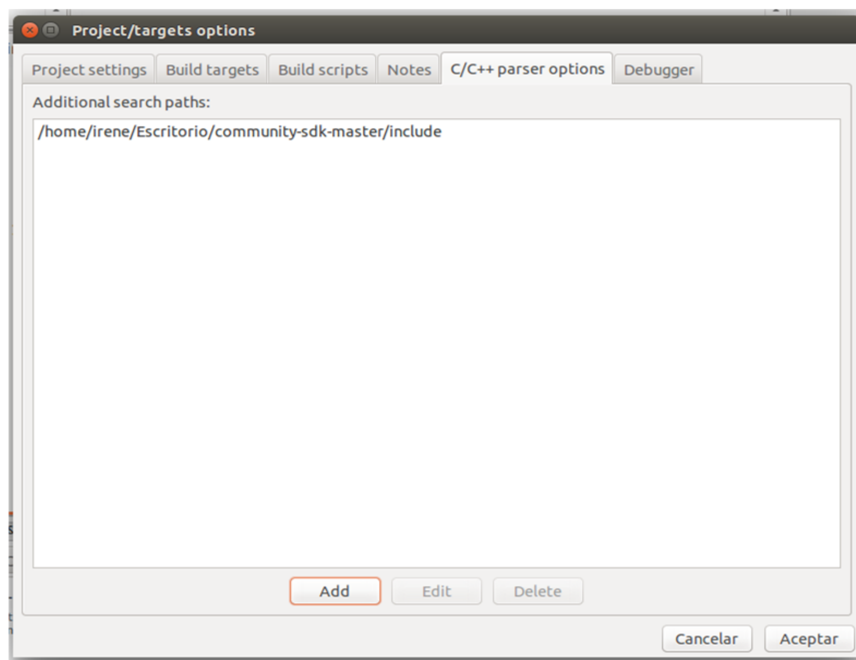


Figura 5.3 Añadir ruta de include en Codeblocks

Por otro lado, en Project/Properties…/Project settings/Project's build options… se marca el -std=c++11 tal y como se muestra en la Figura 5.4.

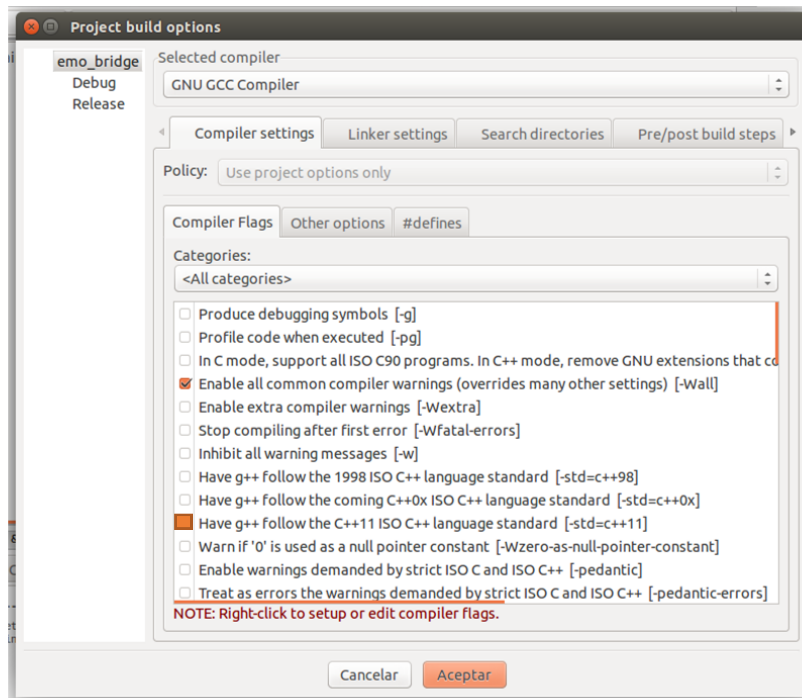


Figura 5.4 Añadir C++11 en Codeblocks

En esa misma ventana, en "linker settings", se añade la ruta de las librerías que se muestra en la Figura 5.5. También se puede añadir, de forma independiente, GLU, GL y glut si se desea poder compilar desde este IDE los ejemplos que usan opengl.

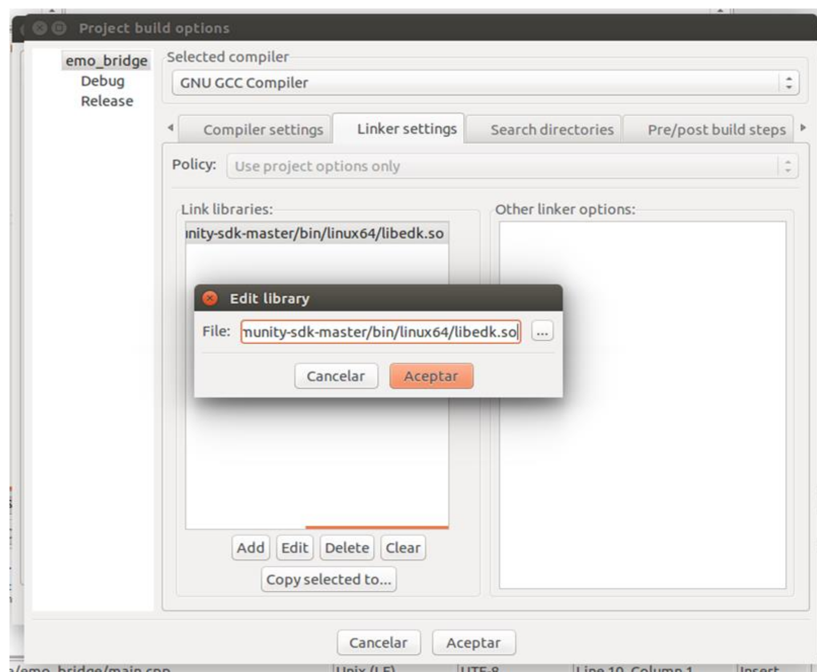


Figura 5.5 Añadir ruta de librerías en Codeblocks

Por último y también en esa misma ventana, en "Search directories", se añade la ruta de los "includes" que se muestra en la Figura 5.6 para que el compilador encuentre los archivos.

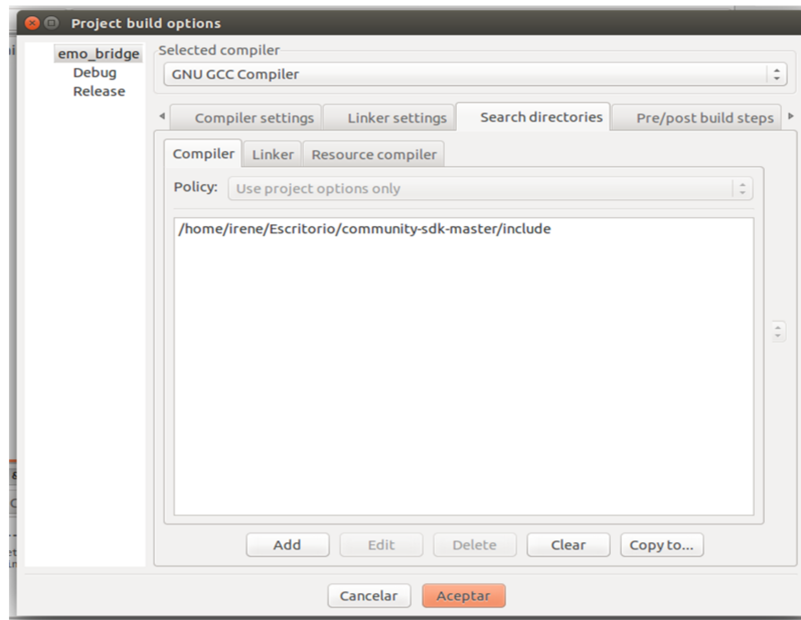


Figura 5.6 Añadir ruta de directorios en Codeblocks

Tras realizar este proceso, el IDE Codeblocks está preparado para poder desarrollar fácilmente y compilar *software* de Emotiv.

5.2.5 ROS

Debido a los requerimientos del Kinova Mico[2], se ha instalado ROS Indigo en Ubuntu 14.04. Por otro lado, puesto que el Emotiv requiere de Ubuntu 16.04, hace uso de ROS Kinetic. Este proceso de instalación está detallado en la documentación de ROS (http://wiki.ros.org/indigo/Installation/Ubuntu) y se considera innecesario incluirlo en esta memoria.

Para comenzar a trabajar con ROS es necesario crear un directorio de trabajo o *workspace*. El proceso detallado se puede encontrar en el Anexo I.

En Ubuntu 14.04, una vez se disponga del *workspace*, se puede instalar el paquete de Kinova (https://github.com/Kinovarobotics/kinova-ros). Esto se hace desde terminal mediante los siguientes comandos:

```
$ cd catkin_ws/src
$ git clone https://github.com/Kinovarobotics/kinova-
ros.git kinova-ros
$ cd ..
$ catkin_make
```

En Ubuntu 16.04, tras haber creado el directorio de trabajo, basta con añadir el archivo de programación de Emotiv y compilarlo. Para programar dicho archivo se recomienda el uso de Codeblocks para poder hacer uso del autocompletado y los ejemplos del SDK.

5.2.5.1 Configuración del paquete kinova-ros

Al instalar el paquete, dentro de la carpeta ~/catkin_ws/src se encuentra otra carpeta llamada kinova-ros. Dentro, se encuentra una serie de ficheros organizados en subcarpetas por funciones, tal y como se muestra en la Figura 5.7.

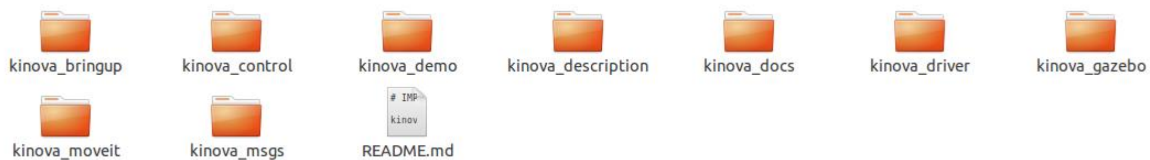


Figura 5.7 Contenido de kinova-ros

- Kinova_bringup: contiene archivos para lanzar de forma sencilla el driver (kinova_driver) y configurar algunos parámetros.
- Kinova_control: incluye el archivo de control que se haya desarrollado.
- Kinova_demo: permite realizar pruebas de funcionamiento fácilmente.
- Kinova_descriptrion: incluye los modelos de mallas de los robots.
- Kinova_docs: se encuentran los documentos de la API de C++ Kinova.
- Kinova_driver: incluye la información avanzada de la API de Kinova. No se recomienda modificar nada sin estar completamente seguro de

que se sabe qué se está haciendo, puesto que es el controlador del robot. Para este proyecto, no ha sido necesario modificarlo.

- Kinova_gazebo: permite lanzar Gazebo y configurar diversos parámetros.
- Kinova_moveit; permite lanzar MoveIt! y configurar diversos parámetros.
- Kinova_msgs: incluye información sobre el formato de los mensajes y los servidores.

En muchos archivos se encuentran ciertos parámetros comunes. En el caso de los archivos kinova_robot.launch, se encuentra como argumento "kinova_robotType". Este se define como un carácter de 8 bits con el siguiente formato [45]:

```
[{j|m|r|c}{1|2}{s|n}{4|6|7}{s|a}{2|3}{0}{0}]
```

- Categoría del robot `{j|m|r|c}`: jaco, mico, roco o customizado.
- Versión `{1|2}`: versión 1 o 2.
- Tipo de muñeca `{s|n}`: esférica o no esférica
- Grados de libertad {4|6|7}: 4, 6 o 7 grados de libertad
- Tipo de robot {s|a}: robot de servicio o de asistencia
- Número de dedos {2|3}: mano con 2 o 3 dedos
- Otras dos posiciones sin definir y reservadas para posibles mejoras

En este caso, el robot que se usa es el modelo Mico^2 de 6 grados de libertad con muñeca no esférica y 3 dedos en la mano. Por tanto, su referencia sería "m2n6s300". Sin embargo, tal y como se indica en la descripción del paquete kinova-ros, para evitar redundancia se han realizado algunos cambios:

- Los modelos urdf de asistencia se han eliminado, por lo que todos los robots tendrán en su referencia una "s" y nunca una "a".
- Para Mico^1 y Mico^2 se ha de usar la versión 1 "m1". Para Jaco^1 y Jaco^2 se ha de usar la versión 2 "j2".

Por tanto, la referencia que se ha usado en el proyecto es “**m1n6s300**”. Esta ha de cambiarse en los parámetros por defecto de los archivos kinova_robot.launch para facilitar el lanzamiento del roslaunch. En caso de que no se haga, habrá que especificar el tipo de robot mediante “kinova_robotType:=m1n6s300”.

Otro parámetro que aparece en los archivos es “use_urdf”, que permite especificar si la solución cinemática se obtiene del modelo URDF. Esta opción es la recomendada y la que aparece por defecto. Si se pone en “false”, la solución cinemática se basa en el procesamiento interno del robot, activando el nodo kinova_tf_updater para publicar *frames* de acuerdo al algoritmo clásico de Denavit Hartenberg [45].

En resumen, los archivos kinova_robot.launch han de tener la configuración en el encabezado como se muestra en la Figura 5.8:

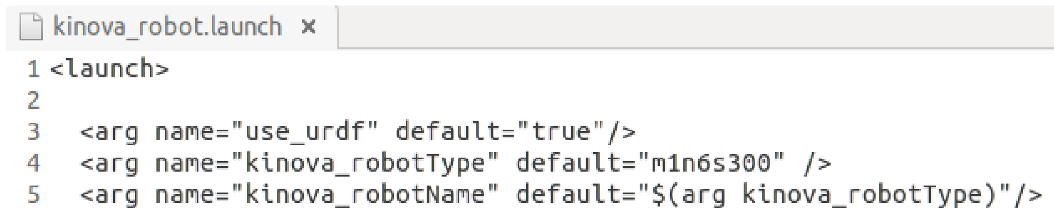


Figura 5.8 Encabezado kinova_robot.launch

Por otro lado, en la carpeta **kinova-control** es necesario realizar varias modificaciones. En el archivo kinova_robot.launch, además de la mostrada en la Figura 5.8, es necesario añadir la descripción de un nodo. Los parámetros se han dejado por defecto a excepción de “type”, donde se ha indicado el nombre del archivo ejecutable que incluye la programación del control.

```
<node name="command_robot_home_pose" pkg="kinova_control" type="goal_pose.py"
      respawn="false" output="screen" args="$(arg kinova_robotType)">
</node>
```

Figura 5.9 Descripción del nodo para kinova-control

Dicho archivo se ha de encontrar en el *workspace* dentro de kinova_control/src y hay que darle permisos de ejecución poniendo en terminal el código que se muestra:

```
$ cd catkin_ws/src/kinova-ros/kinova_control/src
$ chmod +x goal_pose.py
```

Además, en kinova_control hay que modificar también el archivo CMakeLists.txt para incluir el archivo ejecutable. Tras esto, el contenido de dicho archivo será el de la Figura 5.10.

```
CMakeLists.txt ×
1 cmake_minimum_required(VERSION 2.8.3)
2 project(kinova_control)
3 find_package(catkin REQUIRED)
4
5 catkin_package()
6
7 install(DIRECTORY config
8   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION})
9
10 install(DIRECTORY launch
11   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION})
12
13 install(PROGRAMS
14   src/goal_pose.py
15   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
16 )
```

Figura 5.10 Contenido del CMakeLists.txt de kinova-control

La otra carpeta en que se pueden realizar cambios útiles en este proyecto es **kinova-gazebo**. En el archivo robot_launch.launch (Figura 5.11) se han de realizar las mismas modificaciones que se han especificado para kinova_launch.launch. Sin embargo, en este fichero, puesto que configura la simulación, hay otros muchos parámetros de utilidad.

```
robot_launch.launch ×
1 <launch>
2
3   <!-- these are the arguments you can pass this launch file, for example paused:=true -->
4   <arg name="kinova_robotType" default="m1n6s300"/>
5   <arg name="kinova_robotName" default="$(arg kinova_robotType)"/>
6   <arg name="paused" default="false"/>
7   <arg name="use_sim_time" default="true"/>
8   <arg name="gui" default="true"/>
9   <arg name="headless" default="false"/>
10  <arg name="debug" default="false"/>
11  <arg name="use_trajectory_controller" default="true"/>
12  <arg name="is7dof" default="false"/>
13
14  <!-- We resume the logic in empty_world.launch, changing only the name of the world to be launched -->
15  <include file="$(find gazebo_ros)/launch/empty_world.launch">
16    <arg name="world_name" value="$(find kinova_gazebo)/worlds/jaco.world"/>
17    <arg name="debug" value="$(arg debug)" />
18    <arg name="gui" value="$(arg gui)" />
19    <arg name="paused" value="$(arg paused)"/>
20    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
21    <arg name="headless" value="$(arg headless)"/>
22  </include>
23
24  <!-- Load the URDF into the ROS Parameter Server -->
25  <param name="robot_description"
26    command="$(find xacro)/xacro --inorder '$(find kinova_description)/urdf/$(arg kinova_robotType)_standalone.xacro'" />
27
```

Figura 5.11 Configuración de kinova-gazebo

Respecto al primer grupo de parámetros (líneas 3-12), se ha modificado el “kinova_robotType”, el “use_trajectory_controller” y el “is7dof” para que se muestren como en la Figura 5.11. Esto es debido a que “use_trajectory_controller”, cuando está activo, permite realizar el control por posición y trayectoria, mientras que si se desactiva el control se realiza articulación por articulación. Por otro lado, el robot que se controla no tiene 7 grados de libertad, por lo que “is7dof” ha de ser falso.

El segundo grupo de parámetros (líneas 14-23) permite modificar el entorno que tendrá el robot en Gazebo. Es lo que se conoce como mundo. En este caso no se ha modificado puesto que el robot no realiza ningún tipo de interacción con el medio.

Por último, el tercer grupo (líneas 24-26) permite incluir el modelo URDF del robot que se ha especificado con “kinova_robotType”.

Una vez realizadas estas modificaciones, el paquete ROS para trabajar con el brazo robótico está configurado correctamente. Únicamente habría que realizar la programación del archivo que se ha llamado “goal_pose.py” para manejar el Mico[2].

5.2.5.2 Gazebo

La instalación de Gazebo es un proceso sencillo partiendo de ROS. Se han de instalar distintos paquetes para su correcto funcionamiento. Estos son:

```
$ sudo apt-get install ros-indigo-gazebo-ros*
$ sudo apt-get install ros-indigo-gazebo-ros-control
$ sudo apt-get install ros-indigo-ros-controllers*
```

Posteriormente, basta con abrir una terminal y escribir gazebo para ejecutar este programa. También se puede abrir desde el buscador de Ubuntu.

```
$ gazebo
```

5.3 Preparación del *hardware*

A continuación se revisará, de forma breve, la preparación requerida por los dispositivos físicos del proyecto. Cabe recordar que toda la información necesaria para comprender correctamente el funcionamiento de los mismos se desarrolló en los Capítulo 3 y Capítulo 4.

5.3.1 Kinova Mico [2]

En primer lugar, es necesario realizar el ensamblaje de la base y anclar el robot a una mesa en la que tenga espacio suficiente como para moverse hasta su alcance máximo.

Posteriormente, basta con conectar el robot a la alimentación, al joystick y al ordenador mediante los conectores pertinentes que se explicaron en el Capítulo 4.

5.3.2 Emotiv Insight

La preparación del Emotiv es realmente sencilla: basta con realizar la carga de la batería del mismo y, posteriormente, con el casco encendido, conectar el USB Bluetooh a un puerto USB del ordenador para que se empareje con el casco. Toda la información necesaria para realizar este proceso correctamente se encuentra en el Capítulo 3.

5.4 Puesta en marcha

Por último, tras haber realizado la preparación del *hardware* y haber configurado el *software*, se puede proceder a poner en marcha el Emotiv Insight y el Kinova, cada uno de forma independiente. Será una vez realizada la comunicación entre ambos cuando se detallará el funcionamiento conjunto de la plataforma.

5.4.1 Kinova Mico [2]

La puesta en marcha del Kinova es sencilla si se ha realizado previamente la preparación del *hardware* y del *software*. Basta con abrir una terminal y poner el siguiente comando:

```
$ connectrobot
```

Si todo está preparado para el correcto funcionamiento del brazo robótico, se solicitará la contraseña al usuario de Ubuntu y se mostrará un mensaje como el de la Figura 5.12.

```
irene@irene-P850:~$ connectrobot
[sudo] password for irene:
Device connected
```

Figura 5.12 Conexión de Kinova al ordenador

Una vez se ha efectuado la conexión, se puede ejecutar el roslaunch tal y como se muestra:

```
$ roslaunch kinova_bringup kinova_robot.launch
```

En caso de que no se hayan realizado las modificaciones indicadas en el apartado 5.2.5.1, se tendrá que utilizar el siguiente:

```
$ roslaunch kinova_bringup kinova_robot.launch
kinova_robotType:=m1n6s300 use_urdf:=true
```

Cabe destacar que para usar ROS se debe abrir una terminal y ejecutar Roscore. Se trata de una colección de nodos y programas que son imprescindibles en cualquier sistema basado en ROS. Siempre ha de haber un Roscore ejecutándose para que los nodos puedan comunicarse. Para lanzarlo basta con poner en la terminal el comando:

```
$ roscore
```

Sin embargo, como en esta ocasión lo que se va a ejecutar tras el Roscore es un Roslaunch, el propio ROS inicia Roscore si detecta que no se ha hecho ya [48].

Tras escribir el comando de Roslaunch, en la terminal se muestra la información de los parámetros y de los nodos, así como aspectos relativos al proceso de ejecución (Figura 5.13).

```
irene@irene-P850:~$ roslaunch kinova_bringup kinova_robot.launch kinova_robotType:=m1n6s300 use_urdf:=true
... logging to /home/irene/.ros/log/230de0fe-654c-11e7-80c9-6036dde6efd8/roslaunch-irene-P850-21205.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://irene-P850:42510/

SUMMARY
========

PARAMETERS
 * /kinova_number_of_robots: 2
 * /kinova_robots: [{'serial': 'PJ00...
 * /m1n6s300_driver/connection_type: USB
 * /m1n6s300_driver/ethernet/local_broadcast_port: 25025
 * /m1n6s300_driver/ethernet/local_cmd_port: 25000
 * /m1n6s300_driver/ethernet/local_machine_IP: 192.168.100.100
 * /m1n6s300_driver/ethernet/subnet_mask: 255.255.255.0
 * /m1n6s300_driver/jointSpeedLimitParameter1: 10
 * /m1n6s300_driver/jointSpeedLimitParameter2: 20
 * /m1n6s300_driver/robot_name: m1n6s300
 * /m1n6s300_driver/robot_type: m1n6s300
 * /m1n6s300_driver/serial_number: not_set
 * /m1n6s300_driver/torque_parameters/publish_torque_with_gravity_compensation: False
 * /m1n6s300_driver/torque_parameters/use_estimated_COM_parameters: False
 * /robot_description: <?xml version="1....
 * /rosdistro: indigo
 * /rosversion: 1.11.21

NODES
  /
    m1n6s300_driver (kinova_driver/kinova_arm_driver)
    m1n6s300_state_publisher (robot_state_publisher/robot_state_publisher)

ROS_MASTER_URI=http://localhost:11311

core service [/rosout] found
process[m1n6s300_driver-1]: started with pid [21226]
process[m1n6s300_state_publisher-2]: started with pid [21227]
[ INFO] [1499677530.774187506]: kinova_robotType is m1n6s300.
```

Figura 5.13 Lanzamiento de Roslaunch

Tras esto, el robot se mueve hasta situarse en su posición *home*. Una forma de comprobar que el robot está funcionando como es debido es hacer uso del kinova-demo para probarlo. Por ejemplo:

```
$ rosrun kinova_demo joints_action_client.py -v -r
m1n6s300 degree -- 0 0 10 0 0 0
```

Con este primer ejemplo, se rota la 3ª articulación 10° y se muestra en la terminal información sobre la nueva posición. Esto se debe a que el comando incluye "-v". Respecto a "-r", indica que el valor que se muestra es relativo y no absoluto.Por último, "degree" indica que el valor de la rotación se da en grados y no en radianes.

```
$ rosrun kinova_demo pose_action_client.py -v -r
m1n6s300 mdeg -- 0.01 0 0 0 10 10
```

Con este otro ejemplo, se moverá el robot 1cm en el eje X y rotará 10° en torno al eje Y y 10° en torno al eje Z. Los parámetros "-v" y "-r" se usan de igual manera al ejemplo anterior. Respecto a mdeg, hace referencia a que las unidades dadas por el comando se refieren a metros y grados.

Cabe añadir cuál es el sistema de coordenadas que sigue el robot, de forma que se puedan elegir los movimientos adecuadamente para evitar colisiones.

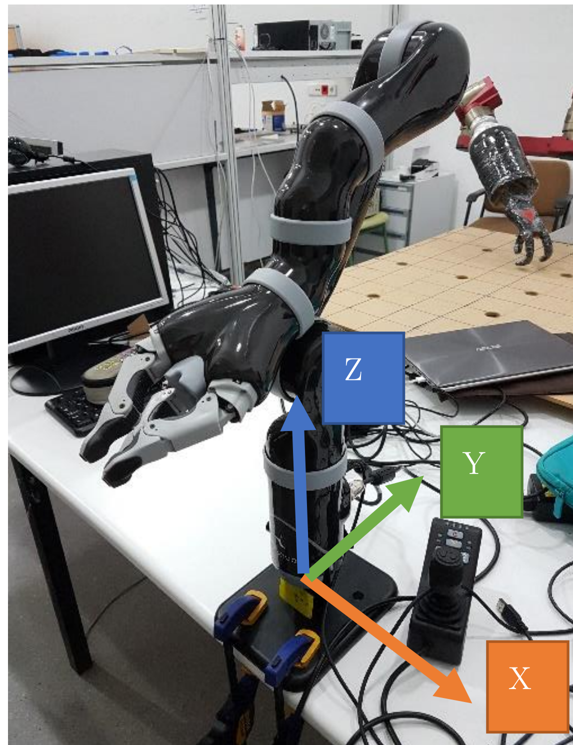


Figura 5.14 Sistema de coordenadas del brazo robótico

También es importante saber qué orden siguen las articulaciones para las funciones en que se refieren los movimientos articulación por articulación.

- 1ª articulación: base
- 2ª articulación: hombro
- 3ª articulación: codo
- 4ª articulación: primera muñeca
- 5ª articulación: segunda muñeca
- 6ª articulación: mano

5.4.2 Emotiv Insight

La puesta en marcha de Emotiv dependerá de la acción que se quiera realizar con él. Para hacer un entrenamiento o practicar las habilidades de detección, basta con abrir el Emotiv ControlPanel.

Por otro lado, si se quiere probar algún ejemplo del SDK se pueden ejecutar tal y como se ha explicado en el apartado 5.2.2.1.

Por último, si se quiere compilar una aplicación desarrollada por el propio usuario, podrá usar el IDE convenientemente preparado para ello.

Capítulo 6

Software desarrollado

6.1 Introducción

En el presente capítulo se describirá el *software* desarrollado para el proyecto, incluyendo la programación del Kinova, la del Emotiv y la comunicación entre ambos.

En primer lugar, se hará una descripción detallada del funcionamiento de los programas y, para completar el análisis de los mismos, se explicarán determinados fragmentos de código.

Cabe destacar que aquí se incluyen las partes que se consideran de especial relevancia para el entendimiento del proceso, pero el código completo se encuentra en los Anexo II, Anexo III y Anexo IV.

6.2 Kinova Mico²

El programa para controlar el Kinova está desarrollado haciendo uso del paquete kinova-ros. El código con propiedades de ejecución se encuentra dentro de

la carpeta /kinova-control/src y está escrito en Python. La principal razón de haber elegido este lenguaje de programación es que los ejemplos de kinova-demo están escritos en dicho lenguaje, por lo que usar el mismo facilita el entendimiento del código y la posible reutilización de ciertas partes.

Por otro lado, tal y como se anticipó en capítulos previos, ROS permite utilizar Gazebo para probar los programas antes de implementarlos en los robots reales. Sin embargo, en este caso, los *topics* que se crean al utilizar el robot real y el simulado son distintos, por lo que el código no es compatible.

Por esta razón y ante la necesidad de realizar pruebas simuladas para prever los movimientos del robot, se han programado dos códigos para Kinova: uno para el robot real y otro para el simulador.

6.2.1 Simulación

El programa de simulación se desarrolló con el objetivo único de experimentar con el brazo róbotico y conocer sus movimientos y restricciones, así como probar las comunicaciones con Emotiv.

El código, escrito en Python, está formado por un *main* sencillo desde el que se llama a diversas funciones. En la Figura 6.1 se muestra el contenido del *main*.

```
if __name__ == '__main__':
  try:
    rospy.init_node('listener', anonymous=True)
    prefix, nbJoints, nbfingers = argumentParser(None)
    rospy.sleep(1)
    print "Let's move the arm"
    moveJoint ([0.0,2.9,1.3,4.2,1.4,0.0],prefix,nbJoints)
    listener()
  except rospy.ROSInterruptException:
    print "program interrupted before completion"
```

Figura 6.1 Main del programa para Gazebo

En primer lugar, *rospy.init_node('listener', Anonymous=True)* se encarga de hacer saber a Rospy el nombre del nodo, lo que le permite iniciar la comunicación

con el ROS Master. El nombre del nodo es, en este caso, *listener*, puesto que es el encargado de escuchar los datos que le envía el Emotiv. Por otro lado, *Anonymous=True* asegura que el nodo tiene un nombre único al añadir números aleatorios al final. Es una buena costumbre de programación ROS añadirlo, sobre todo cuando se manejan varios nodos.

A continuación, se encuentra la función *prefix, nbJoints, nbfingers = argumentParser(None).* Como se observa, no se pasa ningún argumento. Esto es debido a que el argumento se añade desde la propia función, que se explicará posteriormente.

Tras esto, lo siguiente que se encuentra en el *main* (Figura 6.1), es un *rospy.sleep(1).* Esto da lugar a que ROS permanezca inactivo durante 1 segundo. Se ha incluido con el objetivo de permitir que Gazebo se inicie sin afectar al funcionamiento del programa.

Posteriormente se ha incluido un *print*, que sirve como mensaje de depuración para saber que ROS se ha vuelto a activar tras el *sleep*.

A continuación, se realiza una llamada a la función *moveJoint* con los parámetros *([0.0, 2.9, 1.3, 4.2, 1.4, 0.0], prefix, nbJoints).* Dicha función se encarga de mover el brazo robótico a una posición final tras una serie de rotaciones y traslaciones dadas. En este caso, los movimientos indicados llevan al robot a su posición *home.*

Hasta el momento, se ha conseguido inicializar el nodo ROS y se ha movido el robot a la posición home, desde la que realizará los movimientos. Sin embargo, para iniciar la comunicación, es necesario ejecutar la función *listener()*, que incluye funciones para suscribirse al *topic* de Emotiv y ejecutar la función *callback()* al recibir un mensaje.

Por último, el *main* incluye una estructura *try … except*, que permite mostrar un mensaje de error para avisar de que la ejecución se ha interrumpido. Dicho mensaje aparece en la terminal puesto que hace uso de la función *print().*

A continuación, se muestran los códigos de las funciones que se han llamado desde el *main* y, por tanto, se han nombrado previamente.

ArgumentParser (Figura 6.2): en esta función se observa, tras una breve descripción, que se añade un argumento mediante la función de Python *parser.add_argument.*

```
def argumentParser(argument):
    """ Argument parser """
    parser = argparse.ArgumentParser(description='Drive robot joint to command position')
    parser.add_argument('kinova_robotType', metavar='kinova_robotType', type=str,
                        default='m1n6s300',
                        help='Format: [{j|m|r|c}{1|2}{s|n}{4|6|7}{s|a}{2|3}{0}{0}].')
    argv = rospy.myargv()
    args_ = parser.parse_args(argv[1:])
    prefix = args_.kinova_robotType
    print "Robot type: %s" % prefix
    nbJoints = int(args_.kinova_robotType[3])
    print "Number of joints: %d" % nbJoints
    nbfingers = int(args_.kinova_robotType[5])
    print "Number of fingers: %d" %nbfingers
    return prefix, nbJoints, nbfingers
```

Figura 6.2 Función argumentParser del programa para Gazebo

En este caso, este consta de:

- el nombre, que es 'kinova_robotType' (parámetro de kinova-ros ya explicado en el capítulo anterior).
- el metavar que coincide con el nombre.
- el tipo al que se quiere convertir la línea, en este caso *string.*
- el *default*, que es el valor por defecto que usará la función en caso de que no haya argumento, es m1n6s300, que corresponde al robot que se usa en el proyecto.
- la ayuda, que incluye una breve descripción de cómo se construye el nombre.

A continuación, lo que se hace es extraer la información contenida en el argumento, que es "m1n6s300". Por un lado, se guarda en la variable "prefix" la información completa del argumento, que servirá para los nombres de los *topics.* Además, se almacena en la variable "nbJoints" el valor del número de

articulaciones, en este caso 6 (posición 3 del vector). Por último, se guarda también el dato de número de dedos, que es 3 y está almacenado en la posición 5 del vector.

Finalmente, la función devuelve los parámetros “prefix”, “nbJoints” y “nbFingers”.

MoveJoint (Figura 6.3): es la función encargada de mover las articulaciones. para ello, es necesario publicar los mensajes en los *topics* correspondientes.

```
def moveJoint (jointcmds,prefix,nbJoints):
  topic_name = '/' + prefix + '/effort_joint_trajectory_controller/command'
  pub = rospy.Publisher(topic_name, JointTrajectory, queue_size=1)
  jointCmd = JointTrajectory()
  point = JointTrajectoryPoint()
  jointCmd.header.stamp = rospy.Time.now() + rospy.Duration.from_sec(0.0);
  point.time_from_start = rospy.Duration.from_sec(2.0)
  for i in range(0, nbJoints):
    jointCmd.joint_names.append(prefix +'_joint_'+str(i+1))
    point.positions.append(jointcmds[i])
    point.velocities.append(0)
    point.accelerations.append(0)
    point.effort.append(0)
  jointCmd.points.append(point)
  rate = rospy.Rate(100)
  count = 0
  while (count < 1000):
    pub.publish(jointCmd)
    count = count + 1
    rate.sleep()
```

Figura 6.3 Función moveJoint del programa de Gazebo

En primer lugar, en la función se guarda en la variable “topic_name” el nombre del topic en que se va a publicar. Éste se construye haciendo uso del nombre del robot, que se había almacenado previamente en la variable “prefix”, la cual recibe como parámetro. Cabe recordar que el hecho de que este *topic* no exista en el Kinova real es la razón por la que ha sido necesario crear un código para el simulador y otro para el robot.

A continuación, se crea un publicador de mensajes para dicho *topic* mediante *rospy.Publisher(topic_name, JointTrajectory, queue_size=1)*. Esto significa que los mensajes publicados en el *topic* serán del tipo JointTrajectory, que pertenece a

trajectory_msgs y, además, la cola que almacena los mensajes salientes tendrá tamaño 1.

Posteriormente, se realiza una gestión del tiempo que es importante para la publicación de los mensajes al simulador. Para ello, se usan las funciones *rospy.Time.now()* y *rospy.Duration.from_sec(2.0)*, que se describen a continuación:

- rospy.Time.now(): obtiene el tiempo actual como una instancia de Rospy.Time.
- rospy.Duration.from_sec(): crea una instancia de Duration a partir del valor en segundos especificado entre parámetros.

El objetivo de estas líneas de código es crear una marca de tiempo (*stamp*) para gestionar el controlador y el publicador de mensajes. De este modo, se podrá saber en qué instante se publica el mensaje, además de qué se publica.

A continuación, se realiza la preparación del mensaje que se va a publicar, "jointCmd". Para ello, se añade la información de los nombres de las articulaciones haciendo uso de la variable "prefix" y de un bucle que recorre el número de articulaciones para crearlas todas. Además de los nombres, se incorpora información de posición, velocidad, aceleración y esfuerzo.

Al finalizar el bucle, se añade a "jointCmd" toda la información obtenida en el mismo. Por último, se realiza la publicación del mensaje mediante *pub.publish(jointCmd)*.

Listener (Figura 6.3): en esta función, se inicia el suscriptor, que recibe mensajes por el topic "emo_topic" de tipo entero sin signo de 8 bits. Cuando recibe un mensaje, se ejecuta la función *callback(),* que se explica a continuación.

```
def listener():
    print "Listener starts"
    rospy.Subscriber("emo_topic", UInt8, callback)
    rospy.spin()
```

Figura 6.4 Función listener del programa de Gazebo

Callback (Figura 6.5): es la función que se ejecuta cuando se recibe un mensaje de Emotiv. Los mensajes que son enviados son datos de tipo entero sin signo de 8 bits y, según el valor recibido, se realizará un movimiento u otro.

```
def callback(data):
    print "Data: %d" %data.data
    while data.data !=0:
     if data.data == 1:
        print "Emotiv state: blink -> move to pose 1"
        moveJoint ([0.1,-0.10,0.10,0.0,0.0,0.0],prefix,nbJoints)
        moveFingers ([6400,6400,6400],prefix,nbfingers)
        print "Movement completed"
     elif data.data == 4:
        print "Emotiv state: surprised -> move to pose 2"
        moveJoint ([-0.1,0.10,-0.10,0.0,0.0,1.3],prefix,nbJoints)
        moveFingers ([0,0,0],prefix,nbfingers)
        print "Movement completed"
```

Figura 6.5 Función callback del programa de Gazebo

En primer lugar, la función muestra en la terminal el dato recibido de Emotiv y comprueba su valor. En este código, que está destinado a servir como prueba con el simulador, no se han incluido todos los movimientos que sí se contemplan con el robot real.

Por tanto, si el dato recibido es un 1, que indica que el casco ha detectado un parpadeo, el robot simulado se mueve a la posición llamada "pose 1". Para ello, realiza una llamada a la función *moveJoint*, explicada previamente. Además, también mueve la mano del robot mediante la función *moveFingers*, que se explica a continuación.

Por el contrario, si el dato es un 4 es debido a que la expresión detectada ha sido cara de sorpresa, y el robot se mueve a la posición "pose 2" usando las mismas funciones que para "pose 1" pero con distintos parámetros.

moveFingers (Figura 6.6): es una función análoga a moveJoints, con la única diferencia del *topic* en el que se publica el mensaje.

```
def moveFingers (jointcmds,prefix,nbJoints):
  topic_name = '/' + prefix + '/effort_finger_trajectory_controller/command'
  pub = rospy.Publisher(topic_name, JointTrajectory, queue_size=1)
  jointCmd = JointTrajectory()
  point = JointTrajectoryPoint()
  jointCmd.header.stamp = rospy.Time.now() + rospy.Duration.from_sec(0.0);
  point.time_from_start = rospy.Duration.from_sec(5.0)
  for i in range(0, nbJoints):
    jointCmd.joint_names.append(prefix +'_joint_finger_'+str(i+1))
    point.positions.append(jointcmds[i])
    point.velocities.append(0)
    point.accelerations.append(0)
    point.effort.append(0)
  jointCmd.points.append(point)
  rate = rospy.Rate(100)
  count = 0
  while (count < 1000):
    pub.publish(jointCmd)
    count = count + 1
    rate.sleep()
```

Figura 6.6 Función moveFingers del programa de Gazebo

Por último, para terminar la descripción completa del código, falta añadir las cabeceras que se incluyen. Estas se muestran en la Figura 6.7, y son:

- *#!/usr/bin/env Python*: esta línea es necesario para asegurar que el programa se ejecuta como un *script* Python.
- *import rospy*: es necesario puesto que se está creando un nodo ROS en el programa.
- *from trajectory_msgs.msg import JointTrajectory*: permite usar el tipo de mensaje JointTrajectory en el código.
- *from trajectory_msgs.msg import JointTrajectoryPoint*: permite usar el tipo de mensaje JointTrajectoryPoint en el código.
- *from std_msgs.msg import UInt8:* permite usar el tipo de mensaje UInt8 en el código.
- *import argparse*: permite hacer uso de las funciones que Python ofrece el *parse* y que se utilizan concretamente en la función *argumentParser*.

```
#! /usr/bin/env python
"""Publishes joint trajectory to move robot to given pose"""

import rospy
from trajectory_msgs.msg import JointTrajectory
from trajectory_msgs.msg import JointTrajectoryPoint
from std_msgs.msg import UInt8
import argparse
```

Figura 6.7 Cabecera del programa de Gazebo

6.2.2 Real

En este apartado se realiza la descripción del código en Python desarrollado para el control del Kinova Mico[2]. Está basado en el explicado previamente en el apartado 6.2.1, cuyo uso se limita a Gazebo. Por tanto, las partes que son similares se evitarán con el fin de agilizar la lectura.

El ***main*** (Figura 6.8) es sencillo, con el objetivo único de iniciar el nodo de comunicación, llamar al ArgumentParser y situar al robot en su posición *home.* Posteriormente, inicia el listener para mantenerse a la espera de mensajes de Emotiv.

```
if __name__ == '__main__':
    try:
        rospy.init_node('listener', anonymous=True)
        prefix, nbJoints, nbFingers = argumentParser(None)
        homeRobot(prefix)
        gripper_client ([0,0,0],prefix)
        rospy.sleep(1)
        listener()
    except rospy.ROSInterruptException:
        print "program interrupted before completion"
```

Figura 6.8 Main del programa de Kinova

ArgumentParser (Figura 6.9): es una función similar a la del programa de Gazebo. La única diferencia en cuanto a contenido es que, en este caso, la variable "prefix" se forma añadiendo detrás del "kinova_robotType" una "_". Esto es debido a que los *topics* en que se publica para controlar el Kinova incluyen este símbolo detrás del nombre del robot.

```
def argumentParser(argument):
    """ Argument parser """
    parser = argparse.ArgumentParser(description='Drive robot joint to command position')
    parser.add_argument('kinova_robotType', metavar='kinova_robotType', type=str,
                        default='m1n6s300',
                        help='Format: [{j|m|r|c}{1|2}{s|n}{4|6|7}{s|a}{2|3}{0}{0}].')
    args_ = parser.parse_args(argument)
    prefix = args_.kinova_robotType + "_"
    nbJoints = int(args_.kinova_robotType[3])
    nbFingers = int(args_.kinova_robotType[5])
    print "Kinova Mico %s, joint: %d, fingers: %d " %(prefix,nbJoints,nbFingers)
    return prefix, nbJoints, nbFingers
```

Figura 6.9 Función argumentParser() del programa de Kinova

HomeRobot (Figura 6.10): se trata de una función que hace uso de un *topic* que permite mover el robot a la posición *home*. Tal y como se mencionó, los *topics* del simulador no son iguales a los del robot real, y este es una muestra de ello. En este caso, se ha hecho uso de un servicio existente para facilitar la tarea de mover el robot a su posición de partida, previa a cada movimiento.

```
def homeRobot(prefix):
    service_address = '/' + prefix + 'driver/in/home_arm'
    rospy.wait_for_service(service_address)
        try:
            print"Let's go home"
            home = rospy.ServiceProxy(service_address, HomeArm)
            home()
            return None
        except rospy.ServiceException, e:
       print "Service call failed: %s"%e
```

Figura 6.10 Función homeRobot() del programa de Kinova

En primer lugar, se guarda en una variable el nombre del servicio y, posteriormente, se usa la función *rospy.wait_for_service()* para esperar hasta dicho servicio esté disponible.

Una vez quede libre el servicio para mover el robot a su posición *home*, se crea el *Proxy* al que se le pasa la dirección del servicio y el método de inicialización, en este caso "HomeArm". Tras esto, basta con llamar a la función *home()*, que es el nombre que se le ha dado al *Proxy*, para que se efectúe el movimiento del robot.

Cartesian Pose Cliente (Figura 6.11): es la función análoga a *moveRobot()* del programa de Gazebo. Sin embargo, tal y como se anticipaba, no es similar a la usada para el simulador.

En este caso, los mensajes que se envían son de tipo *std_msgs* y *geometry_msgs*. Para ello, se usan vectores de tamaño 3 para la posición (x y z) y de tamaño 4 para la orientación, que viene dada en forma de cuaternios.

Sin embargo, para que el usuario no tenga que introducir los cuaternios, que son más difíciles de visualizar, la función acepta la orientación en ángulos de Euler y la transforma en cuaternios.

Posteriormente, se publica la posición y la orientación del robot en la dirección especificada que corresponde con el *topic* de Kinova.

```
def cartesian_pose_client(position, orientation, prefix):
    action_address = '/' + prefix + 'driver/pose_action/tool_pose'
    client = actionlib.SimpleActionClient(action_address, kinova_msgs.msg.ArmPoseAction)
    client.wait_for_server()

    goal = kinova_msgs.msg.ArmPoseGoal()
    goal.pose.header = std_msgs.msg.Header(frame_id=(prefix + 'link_base'))
    goal.pose.pose.position = geometry_msgs.msg.Point(
        x=position[0], y=position[1], z=position[2])
    orientation=EulerXYZ2Quaternion(orientation)
    goal.pose.pose.orientation = geometry_msgs.msg.Quaternion(
        x=orientation[0], y=orientation[1], z=orientation[2], w=orientation[3])
    print('goal.pose arm: {}'.format(goal.pose.pose))
    client.send_goal(goal)

    if client.wait_for_result(rospy.Duration(200.0)):
        return client.get_result()
    else:
        client.cancel_all_goals()
    return None
```

Figura 6.11 Función cartesian_pose_client() del programa de Kinova

Gripper client (Figura 6.12): es la función análoga a *moveFingers()* del programa de Gazebo. Su funcionamiento en cuanto a publicación de la posición es similar a la función *cartesian_pose_client*, puesto que se guarda la dirección del *topic* para posteriormente publicar en él un vector de posición (dedo1 dedo2 dedo3).

```python
def gripper_client(finger_positions, prefix):
    action_address = '/' + prefix + 'driver/fingers_action/finger_positions'
    client = actionlib.SimpleActionClient(action_address,
                                          kinova_msgs.msg.SetFingersPositionAction)
    client.wait_for_server()

    goal = kinova_msgs.msg.SetFingersPositionGoal()
    goal.fingers.finger1 = float(finger_positions[0])
    goal.fingers.finger2 = float(finger_positions[1])
    goal.fingers.finger3 = float(finger_positions[2])
    client.send_goal(goal)

    if client.wait_for_result(rospy.Duration(50.0)):
        return client.get_result()
    else:
        client.cancel_all_goals()
        rospy.WARN('the gripper action timed-out')
        return None
```

Figura 6.12 Función gripper_client() del programa de Kinova

QuaternionNorm y ***EulerXYZ2Quaterinion*** (Figura 6.13): son dos funciones que contienen los cálculos necesarios para normalizar un cuaternio y para convertir un ángulo de Euler en un cuaternio. Se usan para evitar introducir los cálculos en las funciones del funcionamiento del robot.

```python
def QuaternionNorm(Q_raw):
    qx_temp,qy_temp,qz_temp,qw_temp = Q_raw[0:4]
    qnorm = math.sqrt(qx_temp*qx_temp + qy_temp*qy_temp + qz_temp*qz_temp + qw_temp*qw_temp)
    qx_ = qx_temp/qnorm
    qy_ = qy_temp/qnorm
    qz_ = qz_temp/qnorm
    qw_ = qw_temp/qnorm
    Q_normed_ = [qx_, qy_, qz_, qw_]
    return Q_normed_

def EulerXYZ2Quaternion(EulerXYZ_):
    tx_, ty_, tz_ = EulerXYZ_[0:3]
    sx = math.sin(0.5 * tx_)
    cx = math.cos(0.5 * tx_)
    sy = math.sin(0.5 * ty_)
    cy = math.cos(0.5 * ty_)
    sz = math.sin(0.5 * tz_)
    cz = math.cos(0.5 * tz_)

    qx_ = sx * cy * cz + cx * sy * sz
    qy_ = -sx * cy * sz + cx * sy * cz
    qz_ = sx * sy * cz + cx * cy * sz
    qw_ = -sx * sy * sz + cx * cy * cz

    Q_ = [qx_, qy_, qz_, qw_]
    Q_n=QuaternionNorm(Q_)
    return Q_n
```

Figura 6.13 Operaciones con cuaternios en programa de Kinova

Además, al igual que en el programa de Gazebo, se encuentran las funciones *listener* y *callback*. En este caso, ambas funciones son idénticas, con la única diferencia de la forma en que se proporciona la posición de destino, que viene determinada por la función *goal_pose_client*. A modo de ejemplo, se muestra un fragmento de *callback* en la Figura 6.14.

```
def callback(data):
    print "Data: %d" %data.data
    if data.data == 1:
     homeRobot(prefix)
     print "Emotiv state: blink -> move to pose 1"
     cartesian_pose_client ([0.5,-0.30,0.30],[0,0,0],prefix)
     gripper_client ([6400,0,0],prefix)
     print "Movement completed"
```

Figura 6.14 Fragmento de callback de programa de Kinova

Por último, cabe mencionar las cabeceras que es necesario incluir para el correcto funcionamiento del programa. En este caso, además de las mostradas en la Figura 6.7, se han añadido las que se observan en la Figura 6.15.

```
import actionlib
import kinova_msgs.msg
import geometry_msgs.msg
import tf
import std_msgs.msg
import math
import thread
from kinova_msgs.srv import *
from sensor_msgs.msg import JointState
```

Figura 6.15 Cabeceras del programa de Kinova

Destaca la presencia de:

- *kinova_msgs*, *geometry_msgs* y *std_msgs*, que son la principal diferencia en cuanto a los mensajes y la razón por la cual el programa de Gazebo no es válido
- *math,* la librería matemática para las operaciones con cuaternios.
- *actionlib,* para la comunicación cliente-servidor.

6.3 Emotiv Insight

El programa desarrollado para el uso del casco BCI incorpora ciertas funciones del SDK de Emotiv, así como determinados elementos clave de ROS. Su objetivo principal es gestionar la información recibida del casco y enviar mensajes a través de ROS al programa de Kinova.

Emotiv EmoEngine es una abstracción de la funcionalidad de Emotiv que permite comunicar con el casco de Emotiv, recibir datos del EEG mediante eventos y traducir los resultados de la detección en estructuras más sencillas de utilizar llamadas EmoStates. En la Figura 6.16, se muestra un flujograma de alto nivel del uso de EmoEngine que todas las aplicaciones deben seguir.

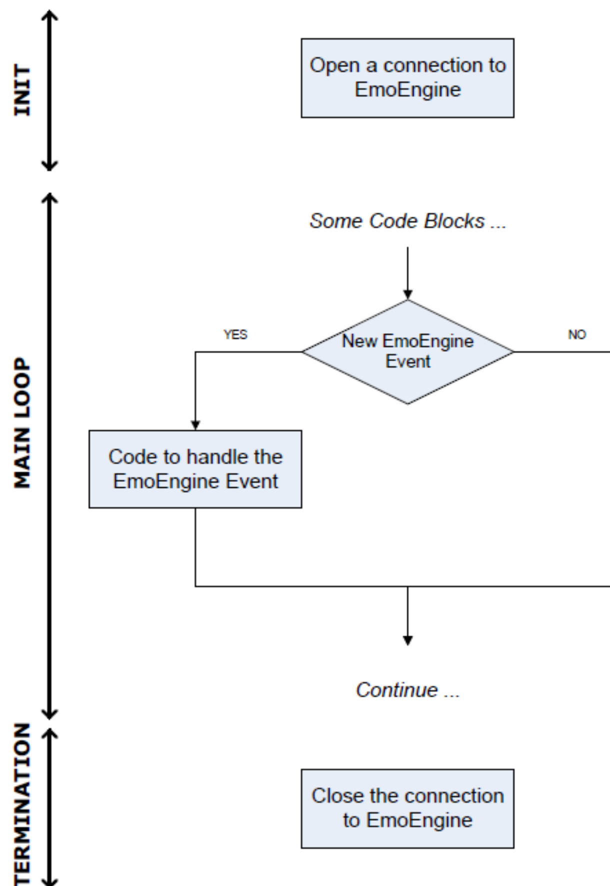


Figura 6.16 Flujograma de funcionamiento de EmoEngine

El programa, escrito en C++, está formado por un *main* y un conjunto de funciones de desarrollo propio que se llaman desde este. Dichas funciones son:

- *logEmoState*: crea un fichero que contiene información sobre el uso de la aplicación: tiempo, calidad del sensor, estado detectado y potencia del mismo…
- *signInEmotiv*: permite acceder al perfil de Emotiv ControlPanel con el que el usuario ha realizado el entrenamiento. Para ello, es necesario conectar con Emotiv Cloud, servidor en el que se almacenan los perfiles de Emotiv.
- *loadProfileEmotiv*: carga el perfil de Emotiv en el que se haya iniciado sesión previamente y obtiene todos los datos del mismo.
- *signal_handler*: permite terminar la ejecución del programa mediante teclado usando Ctrl+c.
- *startConnection*: elige el tipo de conexión a efectuar (casco o simulador), y el tipo de detección a realizar (facial o mental). Además, según lo elegido, se configura la conexión y se realiza.

A continuación, se explican algunos trozos de código que se consideran de especial relevancia para el entendimiento de la aplicación.

6.3.1 Main

El código *main* se compone de distintas partes bien diferenciadas. En primer lugar, se encuentra la declaración de variables. En la Figura 6.17 se muestran las que el usuario debe modificar para configurar su usuario en la aplicación.

```
//User-related variables from Emotiv ControlPanel
std::string userName = "user";
std::string password = "password";
std::string profileName = "PerfilPrueba";

//File-related variables
std::string filename = "emostate_logger.csv";
```

Figura 6.17 Declaración de variables en programa de Emotiv

- *userName*: es el nombre de usuario con el que se inicia sesión en el Emotiv ControlPanel.
- *password*: es la contraseña que se introduce junto al nombre de usuario.
- *profileName*: es el nombre del perfil con el que se ha guardado un entrenamiento asociado a un usuario.
- *filename*: es el nombre del archivo que se genera con la información de las detecciones efectuadas. No es necesario cambiarlo.

A continuación, se inicia ROS y EmoEngine. El código necesario para esto se muestra en la Figura 6.18, donde también se observa la llamada a una de las funciones propias mencionadas previamente. Se trata de startConnection(), que permite realizar la conexión con el dispositivo de Emotiv.

```
//Starting ROS
ros::init(argc, argv, "emo_control");
ros::NodeHandle n;
ros::Publisher emo_pub=n.advertise<std_msgs::UInt8>("emo_topic",1000);
ros::Rate loop_rate(100);

//Starting Emotiv
startConnection();
EmoEngineEventHandle eEvent = IEE_EmoEngineEventCreate();
EmoStateHandle eState = IEE_EmoStateCreate();
std_msgs::UInt8 msg;
```

Figura 6.18 Inicio de ROS y EmoEngine en programa de Emotiv

Respecto a ROS, se inicia el nodo del programa y se crea un gestor de nodos. Además, se crea un publicador de mensajes tipo entero sin signo de 8 bits en el *topic* de nombre “emo_topic”.

Por otro lado, para iniciar Emotiv se llama a la función startConnection(), que se explica en el apartado 6.3.2.1.

Posteriormente, siguiendo con la configuración de Emotiv, se ha de iniciar sesión con las variables definidas en la declaración. Además, se activa la opción de

diagnóstico para asegurar que el funcionamiento es correcto y se llama a la función que permite realizar la detección. Esto se muestra en la Figura 6.19.

```
signInEmotiv(userName, password);
IEE_EnableDiagnostics("log",1,0);
IEE_EmoEngineEventGetEmoState(eEvent, eState);
IS_Init(eState);
```

Figura 6.19 Configuración en programa de Emotiv

De estas funciones, signInEmotiv() es la única que no se encuentra en la librería de Emotiv y que, por tanto, ha sido desarrollada e incluida en el código. Su funcionamiento se muestra en el apartado 6.3.2.3.

Tras iniciar la sesión y activar la detección de EmoStates, el programa inicia un bucle en el que se realiza el resto de acciones. Dicho bucle es un *while* con dos condiciones de permanencia: que ROS se ejecute correctamente y que no se haya pulsado Ctrl+c para salir.

Para poder controlar el paro de ejecución por teclado, se incluye la función signal_handler(), cuya explicación se encuentra en el apartado 6.3.2.2. Por tanto, mientras no se interrumpa este bucle, la ejecución del programa es cíclica. El procedimiento es el siguiente:

1. Se detecta un evento mediante “IEE_EngineGetNextEvent”
2. Se obtiene la información sobre el tipo de evento, que puede ser:
    - Hay un perfil de usuario que cargar en el programa, para lo que se llama a la función loadProfileEmotiv(). El funcionamiento de dicha función se encuentra en el apartado 6.3.2.4.
    - Hay un nuevo EmoState, es decir, se ha realizado una detección por medio de los sensores EEG.
3. En caso de que sea un EmoState la razón del nuevo evento, se inicia el proceso de distinción del comando o expresión (Figura 6.20). Además, se guarda la información para crear el fichero *logger* en la función logEmoState(), explicada en el apartado 6.3.2.5.

```
IEE_EmoEngineEventGetEmoState(eEvent, eState);
float upperFaceAmp = IS_FacialExpressionGetUpperFaceActionPower(eState);
float lowerFaceAmp = IS_FacialExpressionGetLowerFaceActionPower(eState);
float mentalActionAmp = IS_MentalCommandGetCurrentActionPower(eState);
IEE_FacialExpressionAlgo_t upperFaceType = IS_FacialExpressionGetUpperFaceAction(eState);
IEE_FacialExpressionAlgo_t lowerFaceType = IS_FacialExpressionGetLowerFaceAction(eState);
IEE_MentalCommandAction_t mentalActionType = IS_MentalCommandGetCurrentAction(eState);
```

Figura 6.20 Proceso tras detección de EmoState en programa de Emotiv

En primer lugar, se adquiere la información del evento y el estado. Posteriormente, se separa la detección en tres posibles elementos:

- Expresión facial de la zona superior: cejas y ojos.
- Expresión facial de la zona inferior: boca.
- Comando mental.

Se hace uso de las funciones del tipo IS_*Detección*ActionPower para obtener la potencia del comando detectado y guardarlo en variables de tipo *float*. Además, se usan las funciones del tipo IS_*Detección*Action para saber el parámetro detectado de cada tipo y guardarlo en variables de un tipo propio definido por Emotiv.

4. Una vez guardado el tipo de comando detectado y la potencia del mismo, se procede a publicar un mensaje en el "emo_topic". Se muestra como ejemplo, en la Figura 6.21, el código para el comando "sorpresa", siendo análogo para el resto de ellos (tanto faciales como mentales).

```
repetitionCounter_facial++;
switch (upperFaceType)
{
case FE_SURPRISE:
    std::cout << "sorpresa"<< std::endl;
    msg.data = 4;
    now=4;
    break;
```

Figura 6.21 Detección de "sorpresa" en programa de Emotiv

Como se puede observar, el procedimiento es simple: se detecta sorpresa, se imprime por pantalla para que el usuario sepa qué se ha

detectado, y se guarda un valor representativo en la variable msg.data, que es la que se comparte posteriormente por el “emo_topic”.

Por otro lado, se encuentran dos variables no mencionadas previamente: “repetitionCounter_facial” y “now”. El uso de ambas está destinado al control de las detecciones, es decir, se han empleado de forma que hay que detectar cada acción un mínimo de veces antes de que se envíe el mensaje y que no se envíe mensaje de la misma acción dos veces consecutivas.

5. El envío de mensajes se muestra en la Figura 6.22. Se realiza por medio de la publicación de la variable “msg”.

```
if(now!=past){
    std::cout<<"-Facial- Past: "<<past<<"   Now: "<<now<<std::endl;
    past=now;
    repetitionCounter_facial=0;
}
else simpleEvent=false;

if(repetitionCounter_facial==4 || simpleEvent){
    std::cout<<"Publico un nuevo estado facial" <<std::endl;
    emo_pub.publish(msg);
    simpleEvent=false;
}
```

Figura 6.22 Control del envío de mensajes en programa de Emotiv

6. Una vez publicado el mensaje de una detección, el programa vuelve al inicio del bucle, donde realizará el mismo procedimiento cuando se detecte otra acción.

Si se produce algún error o se pulsa Ctrl+C, se sale del bucle. Consecuentemente, se desconecta el EmoEngine (Figura 6.23) para finalizar el programa.

```
IEE_EngineDisconnect();
IEE_EmoStateFree(eState);
IEE_EmoEngineEventFree(eEvent);
```

Figura 6.23 Desconexión de EmoEngine en programa de Emotiv

6.3.2 Funciones

6.3.2.1 startConnection

```
void startConnection(){
```

En primer lugar, esta función espera a que se escriban por teclado dos aspectos de configuración:

- La forma de conexión de Emotiv: se elige entre el *driver* o el simulador.
- El tipo de control: se elige entre expresión facial o comando mental.

Posteriormente, según la opción elegida, se realiza la conexión oportuna, tal y como se observa en la Figura 6.24.

```
case 1:
{
    if (IEE_EngineConnect() != EDK_OK)
    {
        throw std::runtime_error("Emotiv Driver start up failed.");
    }
    break;
}
case 2:
{
    ip = std::string("127.0.0.1");
    std::cout << "La IP es "<< ip << std::endl;
    if (IEE_EngineRemoteConnect(ip.c_str(), composerPort) != EDK_OK)
    {
        std::string errMsg = "Cannot connect to EmoComposer on [" + ip + "]";
        throw std::runtime_error(errMsg.c_str());
    }
    else std::cout << "Composer connected!" << std::endl;
    break;
}
```

Figura 6.24 Conexión en programa de Emotiv

Se observa que la principal diferencia es la función de Emotiv que se usa:

- IEE_EngineConnect: conecta con EmoEngine mediante el controlador de Emotiv.
- IEE_EngineRemoteConnect: conecta con EmoEngine mediante el simulador Emotiv Composer.

Cabe destacar que la dirección IP que se introduce aquí es fija, así como el puerto del Emotiv Composer. Ambos parámetros vienen predeterminados por defecto en la aplicación Emotiv Control Panel con los valores que se han usado en el código.

6.3.2.2 signal_handler

```
void signal_handler(int signal_number)
```

Se trata de una función sencilla que se encuentra definida en los sistemas de tipo Unix. Al pulsar Ctrl+c para detener la ejecución de un programa, se hace un *callback* a esta función y el parámetro "signal_number" adquiere el valor "SIGINT".

Por tanto, se ha incluido en la función (Figura 6.25) una comparación de forma que, cuando se pulse Ctrl+c, la variable "continueProgramme" se vuelva 0, lo que detendrá el bucle de ejecución.

```
void signal_handler(int signal_number)
{
    if(signal_number == SIGINT)
        continueProgramme=false;
}
```

Figura 6.25 Función signal_handler en programa de Emotiv

6.3.2.3 signInEmotiv

```
int signInEmotiv(std::string userName, std::string password)
```

Esta función permite al usuario iniciar sesión con su EmotivID, de forma que pueda usar para la aplicación sus entrenamientos de Emotiv ControlPanel.

Para ello, es necesario conectarse a Emotiv Cloud (EC), un servidor en la nube de Emotiv donde se encuentran almacenados todos los perfiles de usuario que se hayan guardado desde el panel de control. En la Figura 6.26, se observa el proceso a seguir para iniciar sesión.

```
result = EC_Connect();
if(result != EDK_OK)
{
    std::cout << "Cannot connect to Emotiv Cloud" << std::endl;
    connected = false;
}

result = EC_Login(userName.c_str(), password.c_str());
if (result != EDK_OK)
{
    std::cout << "Login attempt failed. Username or password may be incorrect" << std::endl;
    return result;
    connected = false;
}
```

Figura 6.26 Inicio de sesión en programa de Emotiv

En primer lugar, se intenta conectar con EC y, en caso de no poderse, el programa muestra un error.

Seguidamente, se intenta iniciar sesión en EC para acceder a la información de ese perfil.

6.3.2.4 loadProfileEmotiv

```
void loadProfileEmotiv(std::string profileName)
```

Esta función se encarga de, una vez iniciado sesión en Emotiv, cargar el perfil concreto en que se encuentre el entrenamiento realizado. Esto es debido a que un mismo usuario (EmotivID) puede crear varios perfiles que pertenezcan a su misma cuenta, y que contentan distintos entrenamientos.

Por tanto, la forma de proceder de esta función es la siguiente: recibe el nombre del perfil a cargar, busca entre todos los perfiles que contenga el EmotivID

del usuario y lo carga en el programa. Para ello, se usan principalmente dos funciones de Emotiv: "EC_GetProfileId()" y "EC_LoadUserProfile()".

6.3.2.5 logEmoState

```
void logEmoState(std::ostream& os, unsigned int engineUserID,
                 EmoStateHandle eState, bool withHeader)
```

Esta función permite crear un archivo que contiene cierta información de la ejecución de la aplicación: calidad del contacto de cada sensor, acción detectada y su potencia, instante en que se ha detectado y calidad de la conexión del casco.

Para ello se hace uso de funciones propias de Emotiv como IS_GetTimeFromStart() y IS_GetWirelessSignalStatus(), que se muestran en la Figura 6.27.

```
// Log the time stamp and signal status
os << IS_GetTimeFromStart(eState) << ",";
os << static_cast<int>(IS_GetWirelessSignalStatus(eState)) << ",";
```

Figura 6.27 Información del tiempo y la conexión en programa de Emotiv

Además, para comprobar la calidad de los sensores basta con usar la función IS_GetContactQuality(), especificando el canal al que se hace referencia (Figura 6.28).

```
//CQ
os << IS_GetContactQuality(eState, IEE_CHAN_AF3) << ", ";
os << IS_GetContactQuality(eState, IEE_CHAN_AF4) << ", ";
os << IS_GetContactQuality(eState, IEE_CHAN_T7)  << ", ";
os << IS_GetContactQuality(eState, IEE_CHAN_T8)  << ", ";
os << IS_GetContactQuality(eState, IEE_CHAN_Pz)  << ", ";
```

Figura 6.28 Información de calidad del contacto en programa de Emotiv

Para determinar la acción detectada, se hace uso de las mismas funciones que se explican en el código del *main* y que permiten conocer tanto el comando que se ha detectado como la potencia del mismo.

Capítulo 7

Funcionamiento de la aplicación

7.1 Introducción

El presente capítulo incluye la explicación del proceso a realizar para poner en funcionamiento la aplicación. Cabe recordar los elementos que se usan, que se muestran en la Figura 7.1.

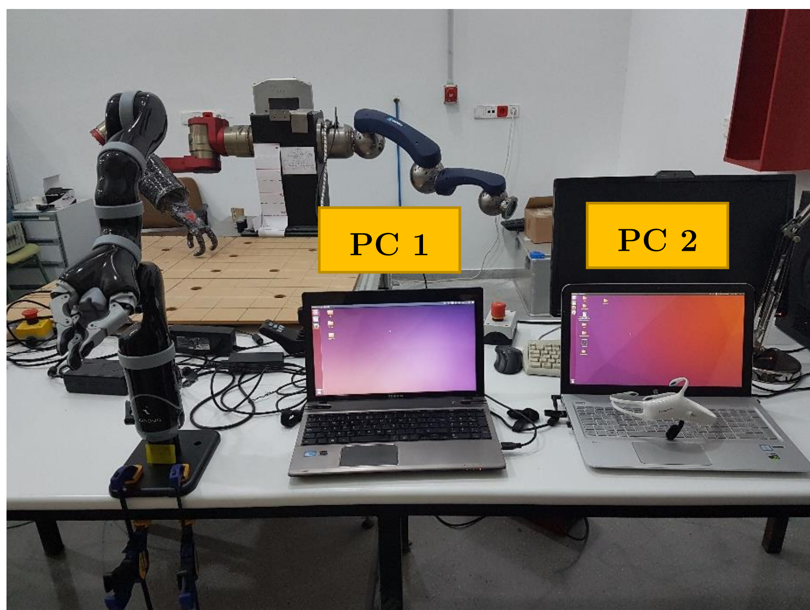


Figura 7.1 Dispositivos de la aplicación

Tal y como se observa en la figura y se ha mencionado a lo largo de esta documentación, se ha hecho uso de dos ordenadores:

- PC1, con Ubuntu 14.04 y ROS Indigo. Se utiliza para controlar el brazo robótico Kinova y actúa como *Listener* en la comunicación.
- PC2, con Ubuntu 16.04 y ROS Kinetic. Se utiliza para conectar con el Emotiv y actúa como *Publisher* o *Talker* en la comunicación.

7.2 Puesta en marcha

A continuación, se enumeran los pasos a seguir para poner en marcha la aplicación mediante la comunicación de los distintos dispositivos:

1. Conectar tanto PC1 como PC2 a la misma red. Para el proyecto se ha utilizado una red creada desde un teléfono móvil, compartiendo la conexión.
2. Encender el brazo robótico desde el interruptor y preparar el PC1 mediante la conexión del USB de Kinova. Tal y como se describió en el Capítulo 5 de preparación del entorno, es necesario conectar el robot y lanzar su driver.

```
$ connectrobot
```

```
irene@irene-P850:~$ connectrobot
[sudo] password for irene:
Device connected
```

Figura 7.2 Establecer conexión con el robot

3. Conocer la dirección IP tanto del PC1 como del PC2. Para ello, se hace uso del comando *ifconfig*. Este programa permite obtener información de la configuración de las interfaces de red que se encuentren en operación. En la Figura 7.3 se muestra la del PC1 y en la Figura 7.4 la del PC2.

```
irene@irene-P850:~$ ifconfig
eth0      Link encap:Ethernet  direcciónHW b8:88:e3:1d:51:aa
          ACTIVO DIFUSIÓN MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Bucle local
          Direc. inet:127.0.0.1  Másc:255.0.0.0
          Dirección inet6: ::1/128 Alcance:Anfitrión
          ACTIVO BUCLE FUNCIONANDO  MTU:65536  Métrica:1
          Paquetes RX:1138 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:1138 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1
          Bytes RX:98153 (98.1 KB)  TX bytes:98153 (98.1 KB)

wlan0     Link encap:Ethernet  direcciónHW 60:36:dd:e6:ef:d8
          Direc. inet:192.168.43.145  Difus.:192.168.43.255  Másc:255.255.255.0
          Dirección inet6: fe80::6236:ddff:fee6:efd8/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:28167 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:18185 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:38630273 (38.6 MB)  TX bytes:2357757 (2.3 MB)
```

Figura 7.3 Dirección IP del PC1

```
irene@irene-HPenvy:~$ ifconfig
enp2s0    Link encap:Ethernet  direcciónHW fc:3f:db:d4:75:21
          ACTIVO DIFUSIÓN MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Bucle local
          Direc. inet:127.0.0.1  Másc:255.0.0.0
          Dirección inet6: ::1/128 Alcance:Anfitrión
          ACTIVO BUCLE FUNCIONANDO  MTU:65536  Métrica:1
          Paquetes RX:99357 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:99357 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:17730514 (17.7 MB)  TX bytes:17730514 (17.7 MB)

wlp3s0    Link encap:Ethernet  direcciónHW 10:02:b5:3f:1d:2b
          Direc. inet:192.168.43.157  Difus.:192.168.43.255  Másc:255.255.255.0
          Dirección inet6: fe80::[illegible]c00a:af64:5638/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
          Paquetes RX:6593 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:5974 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:5964322 (5.9 MB)  TX bytes:613812 (613.8 KB)
```

Figura 7.4 Dirección IP del PC2

Por tanto, las direcciones que se han utilizado en este son las siguientes:

-PC1: 192.168.43.145 *Listener*

-PC2: 192.168.43.157 *Talker*

4. Abrir los archivos .bashrc de ambos ordenadores. Para ello, se puede escribir en terminal el comando que se muestra a continuación.

```
$ gedit ~/.bashrc
```

Otra opción es abrir el archivo manualmente, para lo que hay que hacer Ctrl+h (para mostrar los archivos ocultos) y doble click sobre el mismo.

En las últimas líneas, bajo las variables del estado del sistema que se introdujeron en el Capítulo 5 de preparación del entorno, se debe añadir: export ROS_MASTER_URI=htpp://*IP del Master*:11311. Esto es necesario para que se indique a los nodos ROS dónde pueden encontrar al *Master,* es decir, dónde se va a ejecutar el *Roscore*. En este caso, el *Master* es el *Listener* y su IP es 102.168.43.145, por lo que el comando a añadir queda como se muestra en la Figura 7.5.

```
export ROS_MASTER_URI=http://192.168.43.145:11311
```

Figura 7.5 Variable de entorno ROS_MASTER_URI

Cabe destacar que esto solo será necesario realizarlo cuando se use una red distinta. Es decir, si siempre se usa el mismo dispositivo móvil, las direcciones IP de los dispositivos permanecen iguales y no es necesario cambiarlas.

5. Lanzar *Roscore* en el *Listener*, es decir, en PC1. Para ello, basta con abrir una terminal y escribir el comando.

```
$ roscore
```

En la figura, se muestra la ejecución de *Roscore*, en la que se incluye la información del ROS_MASTER_URI que se ha configurado en el apartado anterior.

```
irene@irene-P850:~$ roscore
... logging to /home/irene/.ros/log/b6c943c8-871c-11e7-af7e-6036dde6efd8/roslaunch-irene-P850-4376.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://irene-P850:42708/
ros_comm version 1.11.21


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.21

NODES

auto-starting new master
process[master]: started with pid [4388]
ROS_MASTER_URI=http://irene-P850:11311/

setting /run_id to b6c943c8-871c-11e7-af7e-6036dde6efd8
process[rosout-1]: started with pid [4401]
started core service [/rosout]
```

Figura 7.6 Ejecución del Roscore

6. Abrir otra terminal en el PC1 (*Listener*) y poner lo siguiente:

```
$ ssh IP_Listener
```

Por tanto, en este caso, el comando es:

```
$ ssh 192.168.43.145
```

Esto permite hacer *login* en la IP del propio ordenador, para lo cual se pide la contraseña. Una vez escrita, la terminal se muestra como en la Figura 7.7.

```
irene@irene-P850:~$ ssh 192.168.43.145
irene@192.168.43.145's password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 4.4.0-83-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

0 packages can be updated.
0 updates are security updates.

New release '16.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2019.
Last login: Mon Jul 17 11:08:44 2017 from 192.168.43.145
irene@irene-P850:~$ netcat -l 1234
```

Figura 7.7 Acceso al ssh Listener

7. Abrir una terminal en el PC2 (*Talker*) y poner lo siguiente:

```
$ ssh IP_Talker
```

Por tanto, en este caso, el comando es:

```
$ ssh 192.168.43.157
```

Al igual que en el *Listener,* esto permite hacer *login* en la IP del propio ordenador, para lo cual se pide la contraseña. Una vez escrita, la terminal del IP2 se muestra como en la Figura 7.8Figura 7.7.

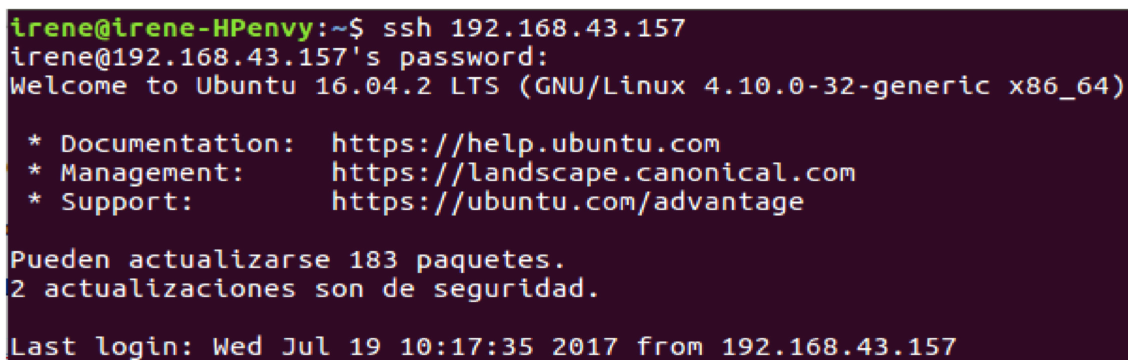


Figura 7.8 Acceso al ssh del Talker

Los puntos 6 y 7 que se han descrito previamente no son necesarios para el funcionamiento de la aplicación. Simplemente sirven para comprobar la conexión entre los dos ordenadores antes de lanzar los programas de ROS. Esto se puede hacer añadiendo en el *Talker* lo siguiente:

```
$ ping IP_Listener
```

Como consecuencia, aparece en la terminal del *Listener* información de la comunicación, de los *bytes* enviados y del instante de tiempo en que se mandan.

Una vez comprobada la comunicación, se puede continuar con el proceso.

8. En el PC1, lanzar el *Roslaunch* del driver de Kinova para ROS, que se encuentra en el paquete kinova-ros. Para ello, se escribe el siguiente comando:

```
$ roslaunch kinova_bringup kinova_robot.launch
kinova_robotType:=m1n6s300 use_urdf:=true
```

9. En el PC1, en otra terminal, se ejecuta el comando que se muestra a continuación. En este caso se trata del ejecutable desarrollado para el control del brazo robótico.

```
$ rosrun kinova_control goal_pose.py m1n6s300
```

Tras esto, comienza la ejecución del código que se ha explicado en el Capítulo 6. El robot se mueve a posición *home* y permanece a la escucha de un mensaje de PC2.

10. En el PC2, conectar el USB Bluetooth, abrir el Emotiv ControlPanel y sincronizar el Emotiv Insight.

11. Abrir una terminal en PC2 y ejecutar la aplicación desarrollada para Emotiv a través de ROS con el siguiente comando:

```
$ rosrun emo_control emo_control
```

Al ejecutarlo, se podrá configurar la conexión tal y hacer uso del programa tal y como se ha detallado en el Capítulo 6. De esta forma, cuando se detecta un nuevo estado de Emotiv, se publica un mensaje en el "emo_topic" y lo recibe la aplicación de Kinova, que se encarga de efectuar el movimiento adecuado.

7.3 Interfaz gráfica

Para facilitar el procedimiento de puesta en marcha de la aplicación, se ha incluido en el proyecto la programación de dos interfaces gráficas. Para ello se ha hecho uso de wxWidgets desde Codeblocks.

Puesto que en la plataforma de experimentación BCI que se desarrolla se usan dos ordenadores distintos (uno para controlar el Kinova y otro para el Emotiv), se han diseñado y programado dos aplicaciones que permiten realizar la puesta en marcha de forma sencilla.

En la Figura 7.9 se muestra la interfaz para Kinova. En primer lugar, como paso previo al uso de la interfaz, se ha de realizar la conexión del robot al ordenador mediante el USB, así como el encendido del mismo.

A continuación, se ha de escribir la dirección IP del *Listener*, que es el propio ordenador en el que se usa el Kinova.

Figura 7.9 Interfaz para Kinova

Al pulsar en el botón "Conectar", se realiza la conexión con el *driver* del robot al introducir la contraseña en la terminal.

Una vez conectado, el botón de "Iniciar ROS" se activa para que se pueda pulsar. Al hacer *click*, se lanza el Roscore con ROS_MASTER_URI en la dirección IP escrita arriba.

Posteriormente, se activa el botón de "Lanzar Kinova", que se encarga de ejecutar el comando que inicia el *driver* del dispositivo en el paquete de ROS.

Por último, se realiza la ejecución del código desarrollado para kinova-control, que se encarga de esperar a recibir datos de Emotiv para efectuar los movimientos.

Por otro lado, en la Figura 7.10 se muestra la interfaz gráfica para controlar el Emotiv.

Figura 7.10 Interfaz para Emotiv

En primer lugar, en la línea superior, hay varios parámetros que configurar:

- Repeticiones para detección: es el número de veces seguidas que se ha de detectar una expresión facial o un comando mental antes de que se mande el mensaje de movimiento al Kinova.
- Dirección IP del *listener*: se trata de la dirección IP del PC1, es decir, del PC en el que se ejecuta el Roscore.
- Usuario y contraseña: se trata de la identificación de Emotiv ControlPanel, es decir, el EmotivID. Se usa para poder acceder a los datos de entrenamiento del usuario.

Posteriormente, se elige el "Método de conexión" con Emotiv, que puede ser:

- Mediante el *driver* del dispositivo. Si se elige esta opción, para poder hacer uso de los entrenamientos se ha de acceder a EmoCloud.
- Mediante el simulador Emotiv Composer.
- Mediante el ControlPanel. Con esta opción, para hacer uso de los entrenamientos se ha de tener abierto el Emotiv ControlPanel con la sesión iniciada.

Además, una vez seleccionado el tipo de conexión a realizar, se ha de elegir el "Tipo de control". En este caso se trata de si se va a usar la detección de expresiones faciales o de comandos mentales.

Una vez configurada la aplicación, se activa el botón "Comenzar". Al darle, se ejecuta el programa desarrollado para el control de Emotiv, que se encarga de publicar mensajes para Kinova según la lectura del EEG.

Una vez finalizado el proceso, se puede terminar la ejecución con el botón "Parar", lo que cierra los programas y abre el fichero que se crea con los datos obtenidos durante la ejecución de la aplicación. Se recuerda que también se puede detener el proceso mediante Ctrl+c.

Sin embargo, el botón "Parar" tiene otro uso, que consiste en abrir el fichero obtenido tras el uso de la aplicación.

Por último, cabe destacar que para ejecutar cualquiera de las interfaces que se ha explicado, hay que abrir una terminal en la carpeta en que se encuentre y poner:

```
$ ./NombreDeLaInterfaz
```

Capítulo 8

Pruebas y resultados finales

8.1 Introducción

En este capítulo se describen las pruebas realizadas en la aplicación, no solo para comprobar el funcionamiento de la misma, sino también para obtener resultados comparables.

Con el objetivo de poder realizar un análisis de las pruebas realizadas, es necesario obtener una realimentación que permita saber si las detecciones se han llevado a cabo de forma adecuada, si el funcionamiento ha sido el correcto, si ha habido pérdida de la calidad del contacto durante la prueba…

Para ello, se ha programado la creación de un archivo (*log*) que permita comparar los experimentos y extraer conclusiones. Se trata de un documento en formato CSV (*Comma-Separated Values*), que se puede abrir con Microsoft Excel o Libreoffice y que incluye los siguientes parámetros:

- Instante de tiempo.
- Calidad de la conexión del casco en cada instante de tiempo.

- Calidad de contacto de cada uno de los sensores en cada instante de tiempo.
- Potencia de detección de todas las acciones en cada instante de tiempo.

En las Figura 8.1 y Figura 8.2 se muestran las dos partes de dicho archivo.

Time	:s Signal	AF3	AF4	T7	T8	Pz
1537.24	2	2	4	4	4	2
1537.72	2	2	4	4	4	2
1537.76	2	4	4	4	4	2
1537.89	2	4	4	4	4	2
1537.99	2	4	4	4	4	2
1538.13	2	4	4	4	4	2
1538.25	2	4	4	4	4	2
1538.39	2	4	4	4	4	2
1538.5	2	4	4	4	4	2
1538.56	2	4	4	4	4	2
1538.73	2	4	4	4	4	2
1538.76	2	4	4	4	4	2
1539.07	2	4	4	4	4	2
1539.15	2	4	4	4	4	2
1539.21	2	4	4	4	4	2
1539.24	2	4	4	4	4	2
1539.39	2	4	4	4	4	2
1539.62	2	4	4	4	4	2
1539.75	2	4	4	4	4	2
1539.78	2	4	4	4	4	2
1540	2	4	4	4	4	2
1540.14	2	4	4	4	4	2
1540.27	2	4	2	4	4	2
1540.3	2	4	2	4	4	2
1540.39	2	4	2	4	4	2
1540.46	2	4	2	4	4	2
1540.5	2	4	2	4	4	2
1540.64	2	4	2	4	4	2

Figura 8.1 Tiempo, calidad de señal y de contacto en el log

Blink	Wink Left	Wink Right	Surprise	Frown	Smile	Clench	Push	Pull	Left	Right
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	0.0387772	0	0	0	0	0	0	0
0	0	0	0.0367806	0	0	0	0	0	0	0
0	0	0	0.0336552	0	0	0	0	0	0	0
0	0	0	0.0189027	0	0	0	0	0	0	0
0	0	0	0.00984651	0	0	0	0	0	0	0
0	0	0	0.0419683	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.197617	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0.0771258	0	0	0	0	0	0
1	0	0	0	0.0771258	0	0	0	0	0	0
1	0	0	0.194794	0	0	0	0	0	0	0
0	0	0	0.194794	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.14515	0	0	0	0	0	0

Figura 8.2 Detección y potencia de la acción en el log

Además, para demostrar el funcionamiento de la aplicación se incluyen, junto a esta memoria, vídeos de determinadas pruebas realizadas.

8.2 Pruebas realizadas

A continuación, se explican las pruebas realizada en la plataforma de experimentación BCI para comprobar su funcionamiento.

8.2.1 Kinova y simulación de Emotiv

La primera prueba consiste en enviar los datos al robot a través del EmoComposer, es decir, haciendo uso del simulador de Emotiv. Por tanto, el objetivo que se ha buscado con este experimento es la comprobación de las comunicaciones con el robot, del movimiento del mismo y del funcionamiento adecuado del código. Se puede encontrar este experimento en los vídeos adjuntos.

Esta prueba estuvo formada de dos fases:

1. Fase de puesta en marcha: se siguió el proceso indicado en el Capítulo 7 para las dos interfaces gráficas. En la de Emotiv, se eligió como método de conexión "Simulador".
2. Fase de comprobación: en esta parte se han marcado los distintos comandos faciales desde el simulador, para comprobar que el robot realizaba los movimientos oportunos.

Las conclusiones extraídas de esta prueba son las siguientes:

- Es muy importante colocarse a una distancia superior al alcance máximo del brazo robótico para evitar colisiones.
- El sistema se puede poner en marcha en un tiempo inferior a 2 minutos.
- El robot realiza los movimientos adecuadamente al recibir los mensajes.
- Si se detecta varias veces seguidas el mismo comando, solo se envía un mensaje para evitar movimientos repetitivos.
- El robot ha de pasar por la posición *home* antes de moverse a otro punto porque las coordenadas son absolutas y el origen es dicha posición.

8.2.2 Kinova y Emotiv

Esta prueba es la comprobación final del funcionamiento de la plataforma BCI desarrollada, usando el Emotiv Insight y el robot Kinova Mico. Por tanto, el objetivo que se ha buscado con este experimento es la comprobación del potencial de la aplicación desarrollada, para conocer el punto en que se encuentra y cómo se podría mejorar. Se puede encontrar en los vídeos adjuntos a esta memoria.

Esta prueba estuvo formada de dos fases:

1. Fase de puesta en marcha: se siguió el proceso indicado en el Capítulo 7 para los dos interfaces gráficas. En la de Emotiv, se eligió como método de conexión "ControlPanel" para poder usar la firma entrenada sin hacer uso de Emotiv Cloud.
2. Fase de comprobación: en esta parte se han realizado una serie de expresiones faciales para analizar el comportamiento del robot ante los mismos.

Las conclusiones extraídas de esta prueba son, además de las comentadas para la simulación, las siguientes:

- Es muy aportante ajustar adecuadamente para cada usuario la sensibilidad de las acciones desde el ControlPanel.
- Es importante ajustar, según la habilidad del usuario, el parámetro de repeticiones hasta detección.
- Es recomendable aprovechar la realimentación que ofrece el avatar del ControlPanel para repetir la expresión o modificarla para mejorar la detección.
- Se debe comprobar periódicamente la conexión del casco para evitar que funcione mal la aplicación.
- Se debe usar un entorno tranquilo, sin ruido y, a ser posible, sin aire que pueda molestar en los ojos.

- Se debe tener en cuenta que no todas las personas pueden realizar el control adecuadamente con el mismo nivel de entrenamiento puesto que, como se ha demostrado en el experimento anterior, el casco se adapta mejor a unos usuarios que a otros.

8.2.3 Entrenamiento y comprobación para 4 sujetos

Para este test, se ha buscado la realización de una comparativa con datos de cuatro sujetos diferentes. Para ello, se ha especificado un proceso a seguir idéntico para los cuatro sujetos.

Este consistía en cuatro fases bien diferenciadas:

1. Fase de acomodación: consistió en poner el casco al sujeto y asegurar un buen contacto de los electrodos. Duración aproximada: inferior a 3 minutos.
2. Fase de experimentación libre: el sujeto probaba sus expresiones faciales con la firma universal. Duración aproximada: 2 minutos.
3. Fase de entrenamiento: se realizó una única sesión de entrenamiento de cada expresión facial disponible. Duración aproximada: 2 minutos.
4. Fase de experimento guiado: el sujeto tuvo que llevar a cabo una serie expresiones faciales, en un orden pautado y especificado previamente. Estas son:
    - Parpadeo
    - Sonrisa
    - Sorpresa
    - Parpadeo
    - Dientes
    - Fruncir
    - Parpadeo

    Duración aproximada: 1 minuto.

Por otro lado, una vez realizado el experimento con los sujetos, se han de extraer una serie de conclusiones. Para ello es necesario comparar cierta información obtenida para los distintos sujetos.

En este caso, se valoran los siguientes aspectos:

- Adaptación del sujeto a la firma universal. Se ha valorado mediante la interpretación propia del sujeto y la realimentación que el ControlPanel ofrece, según las correspondencias conseguidas durante la fase de experimentación libre. Tras ella, se clasifica al sujeto como:

    - "Muy universal": la firma universal funciona bien para la mayoría de acciones faciales.
    - "Medio universal": la firma universal funciona bien para aproximadamente la mitad de las acciones faciales.
    - "Poco universal": la firma universal no funciona bien para prácticamente ninguna acción facial.

- Adaptación del sujeto a su firma entrenada. Se ha valorado mediante el análisis de los archivos *log* obtenidos tras la experimentación. Por tanto, tras dicho análisis, se clasifica el entrenamiento del sujeto como:

    - "Alto rendimiento": la firma entrenada funciona bien para la mayoría de acciones faciales.
    - "Rendimiento medio": la firma entrenada funciona bien para aproximadamente la mitad de las acciones faciales.
    - "Bajo rendimiento": la firma entrenada no funciona bien para prácticamente ninguna acción facial.

	Tras fase 1	Tras fase 4	Comentarios
Sujeto 1	Medio universal	Rendimiento medio	Le cuesta mantener la concentración y no parpadear durante el entrenamiento.
Sujeto 2	Poco universal	Alto rendimiento	Los entrenamientos son muy eficaces.
Sujeto 3	Muy universal	Alto rendimiento	Presenta muy buena adaptación al casco.
Sujeto 4	Poco universal	Bajo rendimiento	El casco nunca ha llegado a hacer buena señal.

Tabla 8.1 Comparativa de sujetos en experimento BCI

Aunque no se trate de una prueba significativa al no incluir una muestra representativa de gente ni distintas variaciones del experimento, se pueden extraer ciertas conclusiones que se podrán o no reafirmar en futuros experimentos:

- No todos los sujetos presentan la misma mejora tras el entrenamiento.
- Por norma general, se consigue mejor rendimiento usando la firma entrenada.
- Hay personas que precisan de un nivel de entrenamiento mucho mayor que otras para que el sistema les detecte sus expresiones.
- La conexión de los electrodos es un parámetro muy influyente en la calidad de la detección.

Capítulo 9

Conclusiones y trabajos futuros

9.1 Conclusiones

Este proyecto se inició con la idea de participar en la investigación del campo del BCI, pues se considera una tecnología de gran potencial en la ingeniería moderna. El objetivo final de esta tecnología es permitir que personas con discapacidades neuromusculares severas puedan comunicarse con su entorno y, por tanto, ganar en independencia y calidad de vida.

Como primera conclusión, se destaca el hecho de haber conseguido alcanzar todos los objetivos planteados en el Capítulo 1. Esto conlleva haber realizado un proceso de investigación, documentación, programación y pruebas largo y, en ocasiones, costoso.

Otro aspecto a tener en cuenta para trabajar con BCI es que hace falta invertir esfuerzo, dinero y tiempo en un campo que todavía es desconocido y no se sabe con certeza si llegará a ser rentable, tal y como se vio al analizar las curvas de expectación de Gartner. Por tanto, se considera importante cualquier pequeña

aportación que un estudiante, un científico o una pequeña empresa pueda hacer para intentar garantizar un futuro al BCI.

Es por esto que se ha considerado necesario que la aplicación desarrollada no sea una más de las tantas que existen, que suponen avances para el BCI pero que no pueden aprovecharse a una escala mayor porque usan *software* cerrado, muy específico o limitado a la aplicación.

Por tanto, un hecho del proyecto que se considera de particular relevancia y que le proporciona un carácter innovador es haber usado ROS para comunicar el dispositivo BCI con el robot. Gracias a esto, la investigación y los programas desarrollados para este proyecto no constituirán una simple aplicación aislada, sino que cualquiera podrá usarlos como base para desarrollar otra más compleja usando el mismo código de Emotiv.

Para ello, previamente se realizó un estudio de la tecnología BCI con el fin de conocer sus bases científicas y trabajar sobre ellas. Se ha estudiado la tecnología existente para conocer la actividad cerebral tanto de forma invasiva como no invasiva. Lógicamente, el proceso de selección de dispositivo BCI para el proyecto se centró en las no invasivas, estudiando un amplio rango de dispositivos EEG y considerando sus pros y sus contras.

Además, para facilitar la puesta en marcha de la aplicación se han diseñado dos interfaces gráficas que evitan el uso de terminales en Linux, que pueden ser complicadas de manejar si no se está familiarizado.

El resultado final es una plataforma para experimentación BCI que permite, por un lado, hacer uso de las aplicaciones de Emotiv para entrenar los comandos mentales y faciales y, por otro lado, emplear estos para controlar un brazo robótico que se mueve a una posición concreta según sea la acción mental detectada.

En conclusión, este TFG se ha centrado en el desarrollo de una aplicación para iniciarse en el trabajo del BCI y, con ello, que garantice el cumplimiento de los objetivos del proyecto. De forma específica y de acuerdo con los resultados

alcanzados tras la realización del mismo, se pueden enumerar las siguientes conclusiones particulares:

- La arquitectura de trabajo propuesta, formada por dos ordenadores con dos versiones distintas de ROS y comunicadas mediante el propio sistema ROS, funciona de forma adecuada. Para comprobarlo se han realizado, en primer lugar, pruebas de envío de datos mediante acceso por SSH, así como mensajes desde el simulador y desde el robot real.

  Como ventajas de la topología utilizada se encuentran la posibilidad de programar por separado y de manera independiente el Emotiv y el Kinova, el no tener que hacer uso de *sockets* para la comunicación, el poder probar cada elemento por separado y la facilidad de conexión.

  Por otro lado, como desventajas se destaca principalmente la necesidad de usar dos ordenadores debido a la incompatibilidad de las versiones de ROS necesarias para Emotiv y Kinova.

- Se ha demostrado que se puede integrar Emotiv con ROS con simplemente crear un ejecutable que funcione con ROS y se compile con *catkin*. Esto se ha probado mediante las pruebas realizadas tanto con el simulador como con el dispositivo real.

- Respecto a la elección de Emotiv, cabe destacar que su funcionamiento general se ha considerado bueno, obteniendo resultados positivos.

  Como ventajas se destaca la abstracción que ofrece, evitando al usuario tener que tratar con las señales, procesarlas y extraer la información de ellas. Además, las funciones son realmente útiles, los electrodos secos hacen, por norma general, buen contacto con el cuero cabelludo y las aplicaciones como ControlPanel y Composer son fáciles de usar y

útiles. Por último, también destaca la actividad en foros respondiendo cuestiones de los usuarios.

Como desventajas se destaca, en primer lugar, la falta de actualización para sistemas Linux y la poca documentación disponible (y de baja calidad).

- Se ha demostrado la utilidad de los simuladores, que permiten realizar pruebas de forma sencilla y evitando tener que hacer uso del *hardware* para ellas.

- Se ha demostrado que se puede iniciar una investigación en el campo del BCI sin necesidad de conocimientos en neurociencia, y que hay multitud de dispositivos disponibles que actúan como capa de abstracción facilitando el desarrollo de aplicaciones.

- Se ha demostrado también que Emotiv ofrece una calidad aceptable para investigación por un precio asequible, lo que fomenta y facilita el desarrollo del BCI.

- Se ha demostrado que se pueden realizar pruebas de corta duración con resultados aceptables para fomentar la divulgación científica en eventos y facilitar la participación de usuarios que muestren interés en el BCI.

- Se han extraído una serie de conclusiones particulares tras la realización de las distintas pruebas, y se encuentran en el Capítulo 8.

Por tanto, de acuerdo con lo expuesto hasta ahora, se extraen las siguientes conclusiones generales:

- No es necesario hacer una gran inversión para trabajar con BCI.

- No es necesario tener conocimientos médicos ni de neuriciencia para trabajar con BCI.
- Emotiv facilita la abstracción de la adquisición y procesado del EEG.
- ROS permite integrar las comunicaciones entre los dos ordenadores.
- ROS facilita la reutilización del código para futuras aplicaciones.
- Se podrá usar la plataforma de experimentación en eventos de índole científica para mostrar el BCI al público.
- Se podrá mejorar la plataforma de experimentación desarrollada de diversas maneras para contribuir en mayor medida al fomento del BCI.

9.2 Trabajos futuros

Este trabajo, al tratarse de una plataforma para experimentación, puede mejorarse y ampliarse en muchos y muy diversos aspectos.

A continuación, se explican algunas de las mejoras aplicables a la aplicación actual:

- Actualizar la versión del SDK de Emotiv, actualmente de la 3.3. a la 3.5. El hecho de que no haya podido actualizar para este proyecto ha provocado que no se haya podido compilar una parte del código. El problema surgió al incorporar al código las funciones necesarias para conectar con Emotiv Cloud e importar un perfil de Emotiv ID. Al compilar el programa con dichas funciones, se mostraba un error que informaba de que no se encontraban determinados elementos, a pesar de que sí se encuentran definidos en las librerías de Emotiv.

  Por esta razón, se abrió un *issue* en el GitHub del SDK de Emotiv, que se muestra en la Figura 9.1. Lo que se comunica es que, al intentar acceder al Emotiv Cloud, aparece el error "*error adding certificate*" y posteriormente no se puede cargar un determinado perfil de entrenamiento en el programa.

La respuesta ofrecida por “tungntEmotiv”, que es una de las personas encargadas del SDK de Emotiv, es que este error se ha solucionado para la versión 3.5 del SDK, pero que todavía no está disponible para Linux.

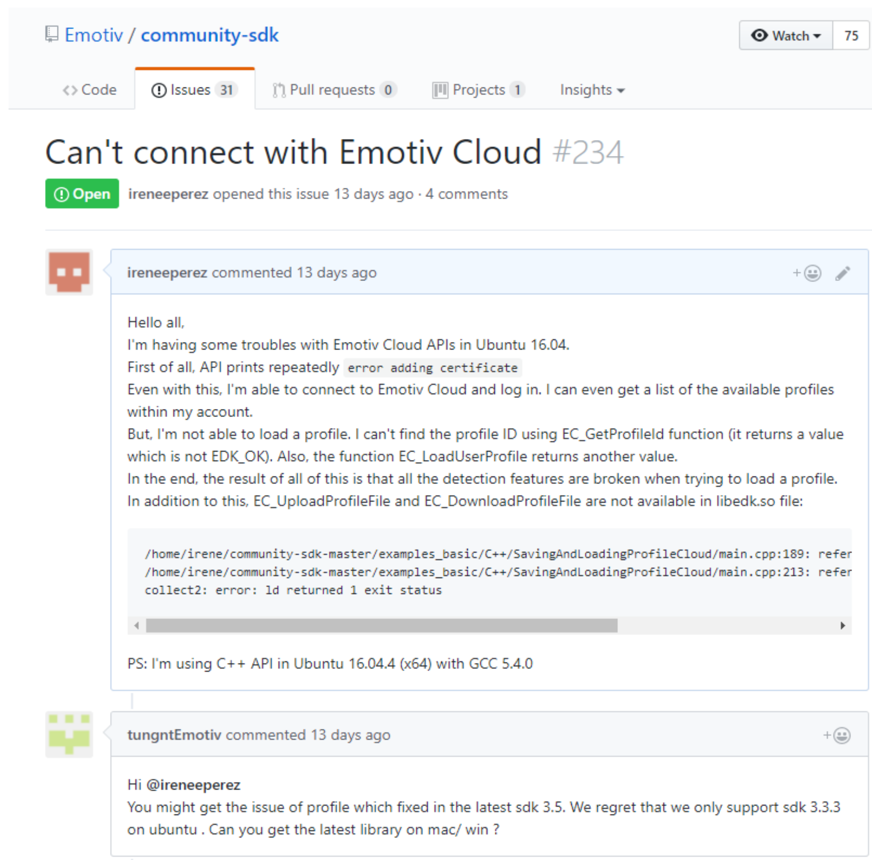


Figura 9.1 Pregunta en GitHub sobre problema del SDK

Por tanto, se deja la posibilidad de añadir el perfil de usuario como trabajo futuro para cuando actualicen la versión del SDK en Ubuntu.

- Modificar el movimiento del robot. En la aplicación actual, la detección de un comando mental envía un mensaje que mueve el robot a una posición asociada con ese comando. La mejora permitiría que el robot se moviera efectuando el movimiento que el usuario está pensando, estableciendo una relación potencia del pensamiento – velocidad.

Por ejemplo, si el usuario piensa derecha progresivamente, el robot se movería a la derecha también de forma gradual.
También se plantea la posibilidad de modificar el movimiento del robot para añadir movimientos más complejos tales como saludar, hacer el símbolo de la paz…

- Portar el driver de Kinova a una versión más reciente de ROS para poder unificar todo el sistema en un único PC.

- Adaptar la plataforma de experimentación a otro robot usando el mismo código de Emotiv para comprobar el funcionamiento en distintas aplicaciones.

- Usar una muestra representativa de personas y realizar un experimento para conocer su adaptación a la firma universal, su capacidad de entrenamiento y su respuesta ante la firma entrenada. Esto permitirá recoger datos para realizar comparaciones entre distintos cascos BCI.

Anexo I

Creación de un espacio de trabajo en ROS

Para poder crear programas que funcionen sobre ROS es necesario crear un *workspace*. Para ello, en primer lugar y tras haber instalado ROS, es necesario ejecutar los siguientes comandos en terminal:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

Al ejecutar estos comandos, lo que se ha hecho es crear una carpeta llamada catkin_ws en el directorio *home*. Además, dentro de catkin_ws se ha creado otra carpeta llamada src.

Mediante *cd* se accede a la carpeta **src** y, desde ahí se ejecuta la inicialización del espacio de trabajo. Como resultado se ha creado dentro de **src** un archivo CMakeLists.txt, que es la herramienta que se encarga de la compilación,

A continuación, es necesario compilar el directorio. Esta acción ha de realizarse desde la raíz del mismo *workspace*, por lo que los comandos a ejecutar son los siguientes:

```
$ cd ..
$ catkin_make
```

Tras esta acción, se crean en el directorio las carpetas **devel** y **build**. De ellos no es necesario realizar modificaciones. La única acción importante es la automatización de las variables de entorno necesarias para ROS. Para ello, se abre el archivo .bashrc con el siguiente comando:

```
$ sudo gedit ~/.bashrc
```

Y al final del mismo se le añaden las rutas de los archivos setup.bash. Para este proyecto eran:

source /opt/ros/kinect/setup.bash

source /home/irene/catkin_ws/devel/setup.bash

Una forma de comprobar que se ha realizado este proceso correctamente es ejecutar el siguiente comando:

```
$ echo $ROS_PACKAGE_PATH
```

La terminal debe mostrar las rutas que se han enlazado. En este caso:

/home/Irene/catkin_ws/src:/opt/ros/kinetic/share

Anexo II

Código de programación de Kinova para Gazebo

```
#Codigo para probar el funcionamiento del Kinova mediante su simulacion
#en Gazebo. No se trata de un código completo ni acabado, simplemente
#es para realizar ciertas pruebas preliminares.

#! /usr/bin/env python
"""Move robot to goal_pose based on Emotiv msg"""

import rospy
import actionlib
import kinova_msgs.msg
import geometry_msgs.msg
import tf
import std_msgs.msg
import math
import thread
from trajectory_msgs.msg import JointTrajectory
from trajectory_msgs.msg import JointTrajectoryPoint
from std_msgs.msg import UInt8
from kinova_msgs.srv import *
```

```
from sensor_msgs.msg import JointState
import argparse

def callback(data):
   #rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
   print "El dato es %d" %data.data
   if data.data == 1:
    homeRobot(prefix)
    print "Emotiv state: blink -> move to pose 1"
    cartesian_pose_client ([0.5,-0.30,0.30],[0,0,0],prefix) #Closer
    gripper_client ([6400,0,0],prefix)
    print "Ya me he movido"
   elif data.data == 2:
    homeRobot(prefix)
    print "Emotiv state: left twink -> move to pose 2"
    cartesian_pose_client ([-0.2,-0.30,0.40],[0,0,0],prefix) #Farther
    gripper_client ([6400,6400,6400],prefix)
    print "Ya me he movido"
   elif data.data == 3:
    homeRobot(prefix)
    print "Emotiv state: right twink -> move to pose 3"
    cartesian_pose_client ([0.0,-0.1,0.80],[0,0,0],prefix) #Up
    gripper_client ([6400,6400,6400],prefix)
    print "Ya me he movido"
   elif data.data == 4:
    homeRobot(prefix)
    print "Emotiv state: surprised -> move to pose 4"
    cartesian_pose_client ([-0.0,-0.60,0.20],[0,0,0],prefix) #Down
    gripper_client ([0,0,0],prefix)
    print "Ya me he movido"

def listener():
    print "Listener starts"
    rospy.Subscriber("emo_topic", UInt8, callback)
    rospy.spin()

def argumentParser(argument):
    """ Argument parser """
```

```
    parser = argparse.ArgumentParser(description='Drive robot joint to 
command position')
    parser.add_argument('kinova_robotType', 
metavar='kinova_robotType', type=str,
                        default='m1n6s300',
                        help='Format: 
[{j|m|r|c}{1|2}{s|n}{4|6|7}{s|a}{2|3}{0}{0}].')
    args_ = parser.parse_args(argument)
    prefix = args_.kinova_robotType + "_"
    nbJoints = int(args_.kinova_robotType[3])
    nbFingers = int(args_.kinova_robotType[5])
    print "Kinova Mico %s, joint: %d, fingers: %d " 
%(prefix,nbJoints,nbFingers)
    return prefix, nbJoints, nbFingers

def QuaternionNorm(Q_raw):
    qx_temp,qy_temp,qz_temp,qw_temp = Q_raw[0:4]
    qnorm = math.sqrt(qx_temp*qx_temp + qy_temp*qy_temp + 
qz_temp*qz_temp + qw_temp*qw_temp)
    qx_ = qx_temp/qnorm
    qy_ = qy_temp/qnorm
    qz_ = qz_temp/qnorm
    qw_ = qw_temp/qnorm
    Q_normed_ = [qx_, qy_, qz_, qw_]
    return Q_normed_

def EulerXYZ2Quaternion(EulerXYZ_):
    tx_, ty_, tz_ = EulerXYZ_[0:3]
    sx = math.sin(0.5 * tx_)
    cx = math.cos(0.5 * tx_)
    sy = math.sin(0.5 * ty_)
    cy = math.cos(0.5 * ty_)
    sz = math.sin(0.5 * tz_)
    cz = math.cos(0.5 * tz_)

    qx_ = sx * cy * cz + cx * sy * sz
    qy_ = -sx * cy * sz + cx * sy * cz
    qz_ = sx * sy * cz + cx * cy * sz
    qw_ = -sx * sy * sz + cx * cy * cz

    Q_ = [qx_, qy_, qz_, qw_]
```

```
    Q_n=QuaternionNorm(Q_)
    return Q_n

def cartesian_pose_client(position, orientation, prefix):
    '''Send a cartesian goal to the action server.'''
    action_address = '/' + prefix + 'driver/pose_action/tool_pose'
    client = actionlib.SimpleActionClient(action_address,
kinova_msgs.msg.ArmPoseAction)
    client.wait_for_server()
    print "Voy a publicar en el topic %s" %action_address
    goal = kinova_msgs.msg.ArmPoseGoal()
    goal.pose.header = std_msgs.msg.Header(frame_id=(prefix +
'link_base'))
    goal.pose.pose.position = geometry_msgs.msg.Point(
        x=position[0], y=position[1], z=position[2])
    orientation=EulerXYZ2Quaternion(orientation)
    goal.pose.pose.orientation = geometry_msgs.msg.Quaternion(
        x=orientation[0], y=orientation[1], z=orientation[2],
w=orientation[3])
    print('goal.pose arm: {}'.format(goal.pose.pose)) # debug
    client.send_goal(goal)
    if client.wait_for_result(rospy.Duration(200.0)):
        print "Publico la posicion final"
        return client.get_result()
    else:
        client.cancel_all_goals()
        print "the cartesian action timed-out"
    return None

def gripper_client(finger_positions, prefix):
    '''Send a gripper goal to the action server.'''
    action_address = '/' + prefix +
'driver/fingers_action/finger_positions'

    client = actionlib.SimpleActionClient(action_address,

kinova_msgs.msg.SetFingersPositionAction)
    client.wait_for_server()

    goal = kinova_msgs.msg.SetFingersPositionGoal()
```

```
    goal.fingers.finger1 = float(finger_positions[0])
    goal.fingers.finger2 = float(finger_positions[1])
    goal.fingers.finger3 = float(finger_positions[2])
    print "Vamos a mover los dedos"
    client.send_goal(goal)
    if client.wait_for_result(rospy.Duration(50.0)):
        return client.get_result()
    else:
        client.cancel_all_goals()
        rospy.WARN('the gripper action timed-out')
        return None

def homeRobot(prefix):
    service_address = '/' + prefix + 'driver/in/home_arm'
    rospy.wait_for_service(service_address)
        try:
           print"Let's go home"
           home = rospy.ServiceProxy(service_address, HomeArm)
           home()
           return None
        except rospy.ServiceException, e:
       print "Service call failed: %s"%e

if __name__ == '__main__':
  try:
    rospy.init_node('listener', anonymous=True)
    prefix, nbJoints, nbFingers = argumentParser(None)
    homeRobot(prefix)
    gripper_client ([0,0,0],prefix)
    rospy.sleep(1)
    listener()
  except rospy.ROSInterruptException:
    print "program interrupted before completion"
```

Anexo III

Código de programación de Kinova

```
#Codigo para probar el funcionamiento del Kinova mediante su simulacion
#en Gazebo. No se trata de un código completo ni acabado, simplemente
#es para realizar ciertas pruebas preliminares.

#! /usr/bin/env python
"""A set of example functions that can be used to control the arm"""
import rospy
import actionlib
import kinova_msgs.msg
import geometry_msgs.msg
import tf
import std_msgs.msg
import math
import thread
from trajectory_msgs.msg import JointTrajectory
from trajectory_msgs.msg import JointTrajectoryPoint
from std_msgs.msg import UInt8
from kinova_msgs.srv import *
from sensor_msgs.msg import JointState
```

```
import argparse

def callback(data):
   #rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
   print "El dato es %d" %data.data
   if data.data == 1:
    #homeRobot(prefix)
    print "Como has parpadeado, muevo los dedos"
    gripper_client ([6400,6400,6400],prefix)
    gripper_client ([0,0,0],prefix)
    print "Ya me he movido"
    rospy.sleep(1.)
   elif data.data == 2:
    #homeRobot(prefix)
    print "Como has guinado el ojo izquierdo, me muevo a la posicion
2"
    gripper_client ([6400,0,0],prefix)
    print "Ya me he movido"
    rospy.sleep(1.)
   elif data.data == 3:
    #homeRobot(prefix)
    print "Como has guinado el ojo derecho, me muevo a la posicion 3"
    gripper_client ([0,0,6400],prefix)
    print "Ya me he movido"
    rospy.sleep(1.)
   elif data.data == 4:
    homeRobot(prefix)
    print "Como te has sorprendido o pensado derecha, me muevo arriba"
    cartesian_pose_client ([0.0,-0.1,0.80],[0,0,0],prefix) #Arriba
    gripper_client ([0,0,0],prefix)
    print "Ya me he movido"
    rospy.sleep(1.)
   elif data.data == 5:
    homeRobot(prefix)
    print "Como has fruncido o empujado, me muevo lejos"
    cartesian_pose_client ([-0.2,-0.30,0.40],[0,0,0],prefix)
#Alejandose de mi
    gripper_client ([0,0,0],prefix)
    print "Ya me he movido"
```

```
    rospy.sleep(1.)
   elif data.data == 6:
    homeRobot(prefix)
    print "Como has sonreido o tirado, me muevo cerca"
    cartesian_pose_client ([0.5,-0.30,0.30],[0,0,0],prefix)
#Acercandose a mi
    gripper_client ([0,0,0],prefix)
    print "Ya me he movido"
    rospy.sleep(1.)
   elif data.data == 7:
    homeRobot(prefix)
    print "Como has pensado izquierda, me muevo izquierda"
    cartesian_pose_client ([-0.0,-0.60,0.20],[0,0,0],prefix) #Abajo
    gripper_client ([0,0,0],prefix)
    print "Ya me he movido"
    rospy.sleep(1.)

def listener():
    print "Entro en listener"
    #rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("emo_topic", UInt8, callback, queue_size=1)
    rospy.spin()

def argumentParser(argument):
        """ Argument parser """
        parser = argparse.ArgumentParser(description='Drive robot
joint to command position')
        parser.add_argument('kinova_robotType',
metavar='kinova_robotType', type=str, default='j2n6a300',
                        help='kinova_RobotType is in format of:
[{j|m|r|c}{1|2}{s|n}{4|6|7}{s|a}{2|3}{0}{0}]. eg: j2n6a300 refers to
jaco v2 6DOF assistive 3fingers. Please be noted that not all options
are valided for different robot types.')
    args_ = parser.parse_args(argument)
    prefix = args_.kinova_robotType + "_"
    nbJoints = int(args_.kinova_robotType[3])
    nbFingers = int(args_.kinova_robotType[5])
    print "El modelo del robot es Kinova Mico %s, que tiene %d
articulaciones y %d dedos" %(prefix,nbJoints,nbFingers)
    return prefix, nbJoints, nbFingers
```

```
def QuaternionNorm(Q_raw):
    qx_temp,qy_temp,qz_temp,qw_temp = Q_raw[0:4]
    qnorm = math.sqrt(qx_temp*qx_temp + qy_temp*qy_temp +
qz_temp*qz_temp + qw_temp*qw_temp)
    qx_ = qx_temp/qnorm
    qy_ = qy_temp/qnorm
    qz_ = qz_temp/qnorm
    qw_ = qw_temp/qnorm
    Q_normed_ = [qx_, qy_, qz_, qw_]
    return Q_normed_

def EulerXYZ2Quaternion(EulerXYZ_):
    tx_, ty_, tz_ = EulerXYZ_[0:3]
    sx = math.sin(0.5 * tx_)
    cx = math.cos(0.5 * tx_)
    sy = math.sin(0.5 * ty_)
    cy = math.cos(0.5 * ty_)
    sz = math.sin(0.5 * tz_)
    cz = math.cos(0.5 * tz_)

    qx_ = sx * cy * cz + cx * sy * sz
    qy_ = -sx * cy * sz + cx * sy * cz
    qz_ = sx * sy * cz + cx * cy * sz
    qw_ = -sx * sy * sz + cx * cy * cz

    Q_ = [qx_, qy_, qz_, qw_]
    Q_n=QuaternionNorm(Q_)
    return Q_n

def cartesian_pose_client(position, orientation, prefix):
    '''Send a cartesian goal to the action server.'''
    action_address = '/' + prefix + 'driver/pose_action/tool_pose'
    client = actionlib.SimpleActionClient(action_address,
kinova_msgs.msg.ArmPoseAction)
    client.wait_for_server()
    print "Voy a publicar en el topic %s" %action_address
    goal = kinova_msgs.msg.ArmPoseGoal()
    goal.pose.header = std_msgs.msg.Header(frame_id=(prefix +
'link_base'))
    goal.pose.pose.position = geometry_msgs.msg.Point(
```

```
        x=position[0], y=position[1], z=position[2])
    orientation=EulerXYZ2Quaternion(orientation)
    goal.pose.pose.orientation = geometry_msgs.msg.Quaternion(
        x=orientation[0], y=orientation[1], z=orientation[2],
w=orientation[3])
    print('goal.pose arm: {}'.format(goal.pose.pose)) # debug
    client.send_goal(goal)
    if client.wait_for_result(rospy.Duration(200.0)):
        print "Publico la posicion final"
        return client.get_result()
    else:
        client.cancel_all_goals()
        print "the cartesian action timed-out"
    return None

def gripper_client(finger_positions, prefix):
    '''Send a gripper goal to the action server.'''
    action_address = '/' + prefix +
'driver/fingers_action/finger_positions'

    client = actionlib.SimpleActionClient(action_address,

kinova_msgs.msg.SetFingersPositionAction)
    client.wait_for_server()

    goal = kinova_msgs.msg.SetFingersPositionGoal()
    goal.fingers.finger1 = float(finger_positions[0])
    goal.fingers.finger2 = float(finger_positions[1])
    goal.fingers.finger3 = float(finger_positions[2])
    print "Vamos a mover los dedos"
    client.send_goal(goal)
    if client.wait_for_result(rospy.Duration(50.0)):
        return client.get_result()
    else:
        client.cancel_all_goals()
        rospy.WARN('the gripper action timed-out')
        return None

def homeRobot(prefix):
    service_address = '/' + prefix + 'driver/in/home_arm'
```

```
    rospy.wait_for_service(service_address)
        try:
           print"Vamos a la posicion home"
           home = rospy.ServiceProxy(service_address, HomeArm)
           home()
           return None
        except rospy.ServiceException, e:
       print "Service call failed: %s"%e

if __name__ == '__main__':
  try:
    rospy.init_node('listener', anonymous=True)
    prefix, nbJoints, nbFingers = argumentParser(None)
    homeRobot(prefix)
    gripper_client ([0,0,0],prefix)
    rospy.sleep(1)
    listener()
  except rospy.ROSInterruptException:
         print "program interrupted before completion"
```

Anexo IV

Código de programación de Emotiv Insight

```
/*Codigo de Emotiv que incluye distintos métodos de conexión y de
control mental. Además, incluye las funciones que no se pueden
utilizar hasta que no actualicen la versión del SDK de Emotiv para
Linux a la 3.5. Cuando se actualice, basta con descomentar dichas
funciones y sus respectivas llamadas. */

#include "ros/ros.h"
#include "std_msgs/UInt8.h"

#include <iostream>
#include <fstream>
#include <map>
#include <sstream>
#include <cassert>
#include <string>
#include <utility>
#include <stdexcept>
#include <cstdlib>
#include <stdio.h>
#include <unistd.h>
```

```
#include <signal.h>

#include <Iedk.h>
#include "IEmoStateDLL.h"
#include <IedkErrorCode.h>
#include <FacialExpressionDetection.h>
#include <MentalCommandDetection.h>
#include "EmotivCloudClient.h"

void logEmoState(std::ostream & os, unsigned int engineUserID,
   EmoStateHandle eState, bool withHeader) {
   // Create the top header
   if (withHeader) {
      os << "Time,";
      os << "Wireless Signal Status,";
      os << "AF3, AF4, T7, T8, Pz,";

      os << "||,";

      os << "Blink,";
      os << "Wink Left,";
      os << "Wink Right,";
      os << "Surprise,";
      os << "Frown,";
      os << "Smile,";
      os << "Clench,";
      os << "Push,";
      os << "Pull,";
      os << "Left,";
      os << "Right,";

      os << std::endl;
   }

   // Log the time stamp and user ID
   os << IS_GetTimeFromStart(eState) << ",";
   os << static_cast < int > (IS_GetWirelessSignalStatus(eState)) <<
",";
```

```
//CQ
os << IS_GetContactQuality(eState, IEE_CHAN_AF3) << ", ";
os << IS_GetContactQuality(eState, IEE_CHAN_AF4) << ", ";
os << IS_GetContactQuality(eState, IEE_CHAN_T7) << ", ";
os << IS_GetContactQuality(eState, IEE_CHAN_T8) << ", ";
os << IS_GetContactQuality(eState, IEE_CHAN_Pz) << ", ";

os << "||,";

// FacialExpression Suite results
os << IS_FacialExpressionIsBlink(eState) << ",";
os << IS_FacialExpressionIsLeftWink(eState) << ",";
os << IS_FacialExpressionIsRightWink(eState) << ",";

std::map < IEE_FacialExpressionAlgo_t, float > expressivStates;
std::map < IEE_MentalCommandAction_t, float > mentalStates;

IEE_FacialExpressionAlgo_t upperFaceAction =
   IS_FacialExpressionGetUpperFaceAction(eState);
float upperFacePower =
IS_FacialExpressionGetUpperFaceActionPower(eState);

IEE_FacialExpressionAlgo_t lowerFaceAction =
   IS_FacialExpressionGetLowerFaceAction(eState);
float lowerFacePower =
IS_FacialExpressionGetLowerFaceActionPower(eState);

IEE_MentalCommandAction_t mentalActionType =
   IS_MentalCommandGetCurrentAction(eState);
float mentalActionPower =
IS_MentalCommandGetCurrentActionPower(eState);

expressivStates[upperFaceAction] = upperFacePower;
expressivStates[lowerFaceAction] = lowerFacePower;
mentalStates[mentalActionType] = mentalActionPower;

os << expressivStates[FE_SURPRISE] << ","; // eyebrow
os << expressivStates[FE_FROWN] << ","; // furrow
os << expressivStates[FE_SMILE] << ","; // smile
os << expressivStates[FE_CLENCH] << ","; // clench
```

```
   os << mentalStates[MC_PULL] << ","; //pull
   os << mentalStates[MC_PUSH] << ","; //push
   os << mentalStates[MC_LEFT] << ","; //left
   os << mentalStates[MC_RIGHT] << ","; //right

   os << std::endl;
   os.flush();
}

/*int signInEmotiv(std::string userName, std::string password)
{
    int result = 0;
    result = EC_Connect();
    if(result != EDK_OK)
    {
        std::cout << "Cannot connect to Emotiv Cloud" << std::endl;
        connected = false;
    }

    result = EC_Login(userName.c_str(), password.c_str());
    if (result != EDK_OK)
    {
        std::cout << "Your login attempt has failed. The username or
password may be incorrect" << std::endl;
        return result;
        connected = false;
    }

    std::cout << "Logged in as " << userName << std::endl;

    result = EC_GetUserDetail(&userCloudID);
    if(result != EDK_OK)
        connected = false;

}

void loadProfileEmotiv(std::string profileName){
    int result = 0;
    int getNumberProfile = EC_GetAllProfileName(userCloudID);
```

```
    std::cout << "Number of profiles: " << getNumberProfile << "\n";

    for (int i = 0; i < getNumberProfile; i++)
    {
        std::cout << "Profile Name: " <<
EC_ProfileNameAtIndex(userCloudID, i) << ", ";

        std::cout << "Profile ID: " <<
EC_ProfileIDAtIndex(userCloudID, i) << ", ";
        std::cout << "Profile type: " <<
                  ((EC_ProfileTypeAtIndex(userCloudID, i) ==
profileFileType::TRAINING) ? "TRAINING" : "EMOKEY") << ", ";

        std::cout << EC_ProfileLastModifiedAtIndex(userCloudID, i) <<
",\r\n";

        result = EC_LoadUserProfile(userCloudID, (int)engineUserID,
EC_ProfileNameAtIndex(userCloudID, i));
        if (result == EDK_OK)
            std::cout << "Loading finished" << std::endl;
        else
            std::cout << "Loading failed" << std::endl;
    }

    int profileID;
    result = EC_GetProfileId(userCloudID, profileName.c_str(),
&profileID);
    if(result!= EDK_OK)
    {
        std::cout << "Profile not found";
    }
    result = EC_LoadUserProfile(userCloudID, (int)engineUserID,
profileID);
    if (result == EDK_OK)
        std::cout << "Loading finished" << std::endl;
    else
        std::cout << "Loading failed" << std::endl;
}*/

void signal_handler(int signal_number) {
   if (signal_number == SIGINT)
```

```
      seguirPrograma = false;
}

void startConnection(char option, char control) {

   const unsigned short composerPort = 1726; //Puerto del Composer
   const unsigned short panelPort = 3008; //Puerto del ControlPanel
   std::string ip;

   std::cout << "Driver: " << option << "\tFacial: " << control <<
std::endl;

   switch (option) {
   case '1':
      {
         if (IEE_EngineConnect() != EDK_OK) {
            throw std::runtime_error("Emotiv Driver start up
failed.");
         }
         break;
      }

   case '2':
      {
         ip = std::string("127.0.0.1");
         std::cout << "La IP es " << ip << std::endl;
         if (IEE_EngineRemoteConnect(ip.c_str(), composerPort) !=
EDK_OK) {
            std::string errMsg = "Cannot connect to EmoComposer on ["
+ ip + "]";
            throw std::runtime_error(errMsg.c_str());
         } else std::cout << "Composer connected!" << std::endl;
         break;
      }
   case '3':
      {
         ip = std::string("127.0.0.1");
         std::cout << "La IP es " << ip << std::endl;
         if (IEE_EngineRemoteConnect(ip.c_str(), panelPort) != EDK_OK)
{
```

```
            std::string errMsg = "Cannot connect to ControlPanel on ["
+ ip + "]";
            throw std::runtime_error(errMsg.c_str());
         } else std::cout << "ControlPanel connected!" << std::endl;
         break;
      }
   default:
      throw std::runtime_error("Invalid option...");
      break;
   }
}

int main(int argc, char * * argv) {
   char option, control;
   int repetition;
   std::string userName, password;

   option = argv[1][0];
   control = argv[2][0];
   userName = argv[3];
   password = argv[4];
   repetition = atoi(argv[5]);

   //std::string profileName = "PerfilPrueba";
   std::string filename = "emostate_logger.csv";

   seguirPrograma = true;

   startConnection(option, control);

   ros::init(argc, argv, "emo_control");
   ros::NodeHandle n;
   ros::Publisher emo_pub = n.advertise < std_msgs::UInt8 >
("emo_topic", 1);
   ros::Rate loop_rate(100);

   bool withHeader = true;
   std::ofstream ofs(filename.c_str());

   EmoEngineEventHandle eEvent = IEE_EmoEngineEventCreate();
```

```
   EmoStateHandle eState = IEE_EmoStateCreate();
   std_msgs::UInt8 msg;

   while (ros::ok() && seguirPrograma) {
      try {
         //signInEmotiv(userName, password);
         IEE_EnableDiagnostics("log", 1, 0);
         IEE_EmoEngineEventGetEmoState(eEvent, eState);
         IS_Init(eState);

         int repetitionCounter_facial;
         int repetitionCounter_mental;
         bool simpleEvent;

         while (ros::ok() && seguirPrograma) {
            int state = IEE_EngineGetNextEvent(eEvent);
            if (state == EDK_OK) {
               IEE_Event_t eventType =
IEE_EmoEngineEventGetType(eEvent);
               IEE_EmoEngineEventGetUserId(eEvent, & engineUserID);
               switch (eventType) {
                  /*case IEE_UserAdded:
                  {
                      if(connected)
                      {
                          loadProfileEmotiv(profileName);
                      }
                      break;
                  }*/
                  // if EmoState has been updated
               case IEE_EmoStateUpdated:
                  {
                     IEE_EmoEngineEventGetEmoState(eEvent, eState);
                     const float timestamp =
IS_GetTimeFromStart(eState);
                     logEmoState(ofs, engineUserID, eState,
withHeader);
                     withHeader = false;
                     float upperFaceAmp =
IS_FacialExpressionGetUpperFaceActionPower(eState);
```

```
                    float lowerFaceAmp =
IS_FacialExpressionGetLowerFaceActionPower(eState);
                    float mentalActionAmp =
IS_MentalCommandGetCurrentActionPower(eState);
                    IEE_FacialExpressionAlgo_t upperFaceType =
IS_FacialExpressionGetUpperFaceAction(eState);
                    IEE_FacialExpressionAlgo_t lowerFaceType =
IS_FacialExpressionGetLowerFaceAction(eState);
                    IEE_MentalCommandAction_t mentalActionType =
IS_MentalCommandGetCurrentAction(eState);
                    //std::cout << timestamp << ": " << "New EmoState
from user " << engineUserID << std::endl;
                    //std::cout << "control es" << control <<
std::endl;
                    if (control == '1') {
                       if (upperFaceAmp > 0.5) {
                          repetitionCounter_facial++;
                          switch (upperFaceType) {
                          case FE_FROWN:
                             std::cout << "fruncido" << std::endl;
                             msg.data = 5;
                             now = 5;
                             break;
                          case FE_SURPRISE:
                             std::cout << "sorpresa" << std::endl;
                             msg.data = 4;
                             now = 4;
                             break;
                          default:
                             break;
                          }
                       } else if (lowerFaceAmp > 0.5) {
                          repetitionCounter_facial++;
                          switch (lowerFaceType) {
                          case FE_SMILE:
                             std::cout << "sonrisa" << std::endl;
                             msg.data = 6;
                             now = 6;
                             break;
                          default:
                             break;
```

```
                                    }
                                } else if
(IS_FacialExpressionIsLeftWink(eState)) {
                                    std::cout << "guiño izdo" << std::endl;
                                    msg.data = 2;
                                    now = 2;
                                    simpleEvent = true;
                                } else if
(IS_FacialExpressionIsRightWink(eState)) {
                                    std::cout << "guiño dcho" << std::endl;
                                    msg.data = 3;
                                    now = 3;
                                    simpleEvent = true;
                                } else if (IS_FacialExpressionIsBlink(eState))
{
                                    std::cout << "parpadeo" << std::endl;
                                    msg.data = 1;
                                    now = 1;
                                    simpleEvent = true;
                                }

                                if (now != past) {
                                    std::cout << "-Facial- Past: " << past << "
Now: " << now << std::endl;
                                    past = now;
                                    repetitionCounter_facial = 0;
                                } else simpleEvent = false;

                                if (repetitionCounter_facial == repetition ||
simpleEvent) {
                                    std::cout << "Publico un nuevo estado
facial" << std::endl;
                                    ros::Duration(1).sleep();
                                    emo_pub.publish(msg);
                                    //repetitionCounter=0;
                                    simpleEvent = false;
                                }
                            } else {
                                if (mentalActionAmp > 0.3) {
                                    repetitionCounter_mental++;
                                    switch (mentalActionType) {
```

```
                        case MC_PULL:
                            std::cout << "tirar" << std::endl;
                            msg.data = 6;
                            now = 6;
                            break;
                        case MC_PUSH:
                            std::cout << "empujar" << std::endl;
                            msg.data = 5;
                            now = 5;
                            break;
                        case MC_LEFT:
                            std::cout << "izquierda" << std::endl;
                            msg.data = 7;
                            now = 7;
                            break;
                        case MC_RIGHT:
                            std::cout << "derecha" << std::endl;
                            msg.data = 4;
                            now = 4;
                            break;
                        default:
                            break;
                        }
                    }
                    if (now != past) {
                        std::cout << "-Mental- Past: " << past << "
Now: " << now << std::endl;
                        past = now;
                        repetitionCounter_mental = 0;
                    }
                    if (repetitionCounter_mental == repetition) {
                        std::cout << "Publico un nuevo estado
mental" << std::endl;
                        emo_pub.publish(msg);
                    }
                }
                break;
            }

        default:
```

```
                    break;
                 }
              } else if (state != EDK_NO_EVENT) {
                 std::cout << "Error" << std::endl;
                 break;
              }

              ros::spinOnce();
              loop_rate.sleep();
           }
        } catch (const std::runtime_error & e) {
           std::cerr << e.what() << std::endl;
           std::cout << "Press 'Enter' to exit..." << std::endl;
           getchar();
          }
   }

   IEE_EngineDisconnect();
   IEE_EmoStateFree(eState);
   IEE_EmoEngineEventFree(eEvent);

   return (1);
}
```

Bibliografía

[1] Estrategias de Inversión, «estrategiasdeinversion.com,» 2017. [En línea]. Available: https://www.estrategiasdeinversion.com/actualidad/noticias/otras/la-robotica-sera-la-mayor-oportunidad-de-inversion-n-361019.

[2] G. Dornhege, J. d. R. Millán, T. Hinterberger, D. J. McFarland y K.-R. Müller, Toward Brain-Computer Interfacing, 2007.

[3] E. Astrand, C. Wardak y S. B. Hamed, «Selective visual attention to drive cognitive brain–machine interfaces: from concepts to neurofeedback and rehabilitation applications,» *Frontiers in Systems Neuroscience,* nº 144, 2014.

[4] J. Wolpaw y E. Wolpaw, Brain-computer interfaces: principles and practice., Oxford University Press, 2012.

[5] B. He, S. Gao, H. Yuan y J. Wolpaw, Neural Engineering, Springer US, 2013.

[6] S. Coyle, T. Ward y C. Markham, «Brain-computer interface using a simplified functional near-infrared spectroscopy system.,» 2007.

[7] Wikipedia, «wikipedia.org,» [En línea]. Available: https://en.wikipedia.org/wiki/Hans_Berger.

[8] Wikipedia, «wikipedia.org,» [En línea]. Available: https://en.wikipedia.org/wiki/Richard_Caton.

[9] Backyard Brains, «backyardbrains.com,» [En línea]. Available: https://backyardbrains.com/experiments/history.

[10] B. Graimann, B. Allison y G. P. Scheller, Brain-computer interfaces: a gentle introduction., Springer, 2010.

[11] E. Fetz, «Operant conditioning of cortical unit activity.,» *Science,* nº 163, 1969.

[12] E. Fetz y D. Finocchio, «Operant conditioning of specific patterns of neural and muscular activity.,» *Science,* nº 174, 1971.

[13] J. Vidal, «Towards direct brain–computer communication.,» 1973.

[14] J. Vidal, «Real-time detection of brain events in EEG.,» 1977.

[15] T. Elbert, B. Rockstroh, W. Lutzenberger y N. Birbaumer, «Biofeedback of slow cortical,» *Electroencephalogr Clin Neurophysiol,* nº 48, 1980.

[16] S. Bozinovski, M.Sestakov y L. Bozinovska, «Using EEG alpha rhythm to control a mobile robot.,» 1988.

[17] J. Wolpaw, D. M. N. Birbaumer, G. Pfurtscheller y T. Vaughan, «Brain-computer interfaces for communication and control.,» *Clin Neurophysiol,* nº 113, 2002.

[18] Gartner, Inc. , «gartner.com,» [En línea]. Available: http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp .

[19] Gartner Inc. , «gartner.com,» 2012. [En línea]. Available: http://www.gartner.com/newsroom/id/2124315.

[20] Gartner Inc. , «gartner.com,» 2013. [En línea]. Available: http://www.gartner.com/newsroom/id/2575515.

[21] Gartner Inc., «gartner.com,» 2014. [En línea]. Available: http://www.gartner.com/newsroom/id/2819918.

[22] Gartner Inc. , «gartner.com,» 2015. [En línea]. Available: http://www.gartner.com/newsroom/id/3114217.

[23] Gartner Inc., «gartner.com,» 2016. [En línea]. Available: http://www.gartner.com/newsroom/id/3412017.

[24] Emotiv Inc. , «emotiv.com,» [En línea]. Available: https://www.emotiv.com/product/emotiv-insight-5-channel-mobile-eeg/.

[25] OpenBCI, «openbci.com,» 2014. [En línea]. Available: http://openbci.com/forum/index.php?p=/discussion/138/buying-dry-active-electrodes.

[26] OpenBCI, «openbci.com,» [En línea]. Available: https://shop.openbci.com/collections/frontpage.

[27] Neurosky, «neurosky.com,» [En línea]. Available: https://store.neurosky.com/pages/mindwave .

[28] Neurosky, «neurosky.com,» [En línea]. Available: https://store.neurosky.com/products/mindset-research-tools .

[29] Olimex, «olimex.com,» [En línea]. Available: https://www.olimex.com/Products/EEG/OpenEEG/EEG-SMT/open-source-hardware.

[30] Interaxon, «choosemuse.com,» [En línea]. Available: http://www.choosemuse.com/developer-kit/.

[31] J. Dearen, «independent.co.uk,» 2016. [En línea]. Available: http://www.independent.co.uk/news/science/drones-brain-thoughts-controlled-bci-brain-computer-interface-brain-controlled-interface-a6996781.html.

[32] T.-H. Hsieh, «robinhsieh.com,» 2013. [En línea]. Available: http://robinhsieh.com/eeg_exoskeleton/.

[33] C. Russomanno, «youtube.com/ConnorRussomanno,» [En línea]. Available: https://www.youtube.com/watch?v=T4OqNtNh_Ls.

[34] J. Corbella, «lavanguardia.com,» 2016. [En línea]. Available: http://www.lavanguardia.com/ciencia/20161013/41977458408/chip-cerebro-tacto-lesion-medular-tetraplejico-pittsburgh.html.

[35] Orca Robotics, «orca-robotics.sourceforge.net,» [En línea]. Available: http://orca-robotics.sourceforge.net/index.html.

[36] YARP, «yarp.it,» [En línea]. Available: http://www.yarp.it/index.html.

[37] MIRA, «mira-project.org,» [En línea]. Available: http://www.mira-project.org/joomla-mira/ .

[38] Wikipedia, «wikipedia.org,» [En línea]. Available: https://en.wikipedia.org/wiki/Robot_Operating_System.

[39] Emotiv Inc., «emotiv.zendesk.com,» [En línea]. Available: https://emotiv.zendesk.com/hc/en-us/articles/205614635-Why-can-t-I-charge-the-Emotiv-Insight-while-in-use-.

[40] Emotiv Inc., «emotiv.zendesk.com,» [En línea]. Available: https://emotiv.zendesk.com/hc/en-us/articles/204821539-I-m-paired-how-do-I-get-my-Insight-sensors-to-make-good-contact-.

[41] Emotiv Inc., «emotiv.com,» [En línea]. Available: https://www.emotiv.com/comparison/ .

[42] «erlerobotics.gitbooks.io,» [En línea]. Available: https://erlerobotics.gitbooks.io/erlerobot/es/ros/ROS.html.

[43] ROS, «wiki.ros.org,» [En línea]. Available: http://wiki.ros.org/ROS/Introduction.

[44] ROS, «wiki.ros.org,» [En línea]. Available: http://wiki.ros.org/ROS/Concepts.

[45] Kinova Robotics, «github.com/Kinovarobotics,» [En línea]. Available: https://github.com/Kinovarobotics/kinova-ros/wiki/README.

[46] Emotiv Inc., «emotiv.com,» [En línea]. Available: https://www.emotiv.com/insight/.

[47] Recipes for Linux, «www.recipesforlinux.com,» [En línea]. Available: http://www.recipesforlinux.com/2011/03/25/trabajando-con-udev-i-introduccion/.

[48] ROS, «wiki.ros.org,» [En línea]. Available: http://wiki.ros.org/roscore .